

AFRL-RI-RS-TR-2010-057
In-House Final Technical Report
February 2010



AGENTS TECHNOLOGY RESEARCH

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

STINFO COPY

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report was cleared for public release by the 88th ABW, Wright-Patterson AFB Public Affairs Office and is available to the general public, including foreign nationals. Copies may be obtained from the Defense Technical Information Center (DTIC) (<http://www.dtic.mil>).

AFRL-RI-RS-TR-2010-057 HAS BEEN REVIEWED AND IS APPROVED FOR
PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION
STATEMENT.

FOR THE DIRECTOR:

/s/
JERRY DUSSAULT
Chief, Advanced Planning Concepts Branch

/s/
JULIE BRICHACEK, Chief
Information Systems Division
Information Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

REPORT DOCUMENTATION PAGE*Form Approved*
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Washington Headquarters Service, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington, DC 20503.

PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.**1. REPORT DATE (DD-MM-YYYY)**
FEBRUARY 2010**2. REPORT TYPE**
Final**3. DATES COVERED (From - To)**
January 2005 – December 2009**4. TITLE AND SUBTITLE**

AGENTS TECHNOLOGY RESEARCH

5a. CONTRACT NUMBER

In House

5b. GRANT NUMBER

N/A

5c. PROGRAM ELEMENT NUMBER

62702F

6. AUTHOR(S)

Robert Wright, Jeffrey Hudack, Nathaniel Gemelli, Steven Loscalzo, and Tsu Kong Lue

5d. PROJECT NUMBER

558S

5e. TASK NUMBER

HA

5f. WORK UNIT NUMBER

TR

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)AFRL/RISC
525 Brooks Road
Rome NY 13441-4505**8. PERFORMING ORGANIZATION
REPORT NUMBER**

N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)AFRL/RISC
525 Brooks Road
Rome NY 13441-4505**10. SPONSOR/MONITOR'S ACRONYM(S)**

N/A

**11. SPONSORING/MONITORING
AGENCY REPORT NUMBER**
AFRL-RI-RS-TR-2010-057**12. DISTRIBUTION AVAILABILITY STATEMENT**

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED. PA# 88ABW-2010-0479 Date Cleared: 16-February-2010

13. SUPPLEMENTARY NOTES**14. ABSTRACT**

This report provides a comprehensive description of three separate efforts pursued by the agents technology research group. The efforts were focused on: state abstraction methods for reinforcement learning, the multi-agent credit assignment problem, and distributed multi-agent reputation management. State abstraction is a technique used to allow machine learning technologies to cope with problems that have large state spaces. This report details the development and analysis of a new algorithm, Reinforcement Learning using State Abstraction via NeuroEvolution (RL-SANE), that utilizes a new technology called neuroevolution to automate the process of state abstraction. The multi-agent credit assignment problem is a situation that arises when multiple learning actors within a domain are only provided with a single global reward signal. Learning is difficult in these scenarios because it is difficult for each agent to determine the value of its contribution to obtaining the global reward. In this report we describe the problem in detail and one specific approach we investigated that uses a Kalman filter to derive local rewards from global rewards. Multi-agent reputation management is important in open domains where the goals or the interests of the agents are diverse and potentially in conflict with one another. Reputation and trust can be used by the agents to determine which other agents in the system it should cooperate with and which it should not. This report details the development of the Affinity Management System (AMS), an approach for managing and learning trust in a distributed fashion that utilizes self-modeling.

15. SUBJECT TERMS

Artificial Intelligence, Software Agents, Multi-Agent Systems, Reinforcement Learning, State Abstractions, Reputation Management, Game Theory

16. SECURITY CLASSIFICATION OF:**17. LIMITATION OF
ABSTRACT****18. NUMBER
OF PAGES****19a. NAME OF RESPONSIBLE PERSON**

Robert Wright

a. REPORT
U**b. ABSTRACT**
U**c. THIS PAGE**
U

UU

57

19b. TELEPHONE NUMBER (Include area code)
N/A

Contents

1	Executive Summary	1
2	Introduction	2
3	Methods	4
3.1	Motivation	4
3.1.1	Reinforcement Learning for Rapid Decision Making in Real-Time Environments	4
3.1.2	Coordinated Multi-Agent Learning	4
3.1.3	Distributed Reputation Management	5
3.2	Background	5
3.2.1	Reinforcement Learning and State Abstraction	6
3.2.2	Multi-Agent Reinforcement Learning	7
3.2.2.1	Related Work	8
3.2.3	Multi-Agent Trust and Reputation	9
3.2.3.1	Defining Trust and Reputation	9
3.2.3.2	Levels of Trust	9
3.3	Approaches	10
3.3.1	RL-SANE	10
3.3.1.1	Related Work	12
3.3.1.2	Bounding of the Abstract State Space	13
3.3.1.3	State Bound Mutation	13
3.3.2	Kalman Filter for Structural Credit Assignment	14
3.3.3	Affinity Management System (AMS)	15
3.3.3.1	Utility in AMS	16
3.3.3.2	Detecting and Penalizing Dishonest Agents	16
3.3.3.3	Probabilistic Cooperation Based on Affinity and Utility	17
3.3.3.4	Related Work	18
4	Results and Analysis	20
4.1	RL-SANE Experiments	20
4.1.1	Benchmark Algorithms	20

4.1.1.1	NeuroEvolution of Augmenting Topologies	20
4.1.1.2	Cerebellar Model Articulation Controller	21
4.1.2	Benchmark Problems and Experimental Setup	22
4.1.2.1	Mountain Car	22
4.1.2.2	Double Inverted Pendulum Balancing	23
4.1.3	Experimental Results	24
4.1.3.1	NEAT Comparison	24
4.1.3.2	CMAC Comparison	27
4.1.3.3	State Bound Mutation Experiments	27
4.2	Kalman Filter Results	28
4.2.1	Test Environment: Grid World	28
4.2.2	Results	29
4.2.3	Agent Scaling	31
4.2.4	Learning Rate	33
4.2.5	Exploration	33
4.2.6	Kalman variance	35
4.3	AMS - Preliminary Results	35
4.3.1	Agent Reputation and Trust testbed	37
4.3.2	Benchmark Agents	38
4.3.3	AMS Benchmark Experiment	39
5	Conclusions	42
5.1	Automated State Abstraction for Reinforcement Learning	42
5.2	Kalman Filter for Multi-Agent Learning	43
5.3	Agent Reputation with Self-modeling	44
6	References	45
7	Symbols, Abbreviations and Acronyms	49

List of Figures

3.1	A graphical depiction of the RL-SANE algorithm	11
4.1	These figures illustrate the two RL benchmark problems, (Left)mountain car and (Right) double inverted pendulum balancing.	23
4.2	This figure shows the structure of typical solution neural networks for the mountain car problem. The β value for RL-SANE was set to 50.	25
4.3	Shows the performance of the RL-SANE algorithm compared to that of NEAT on the mountain car (Left) and the double inverted pendulum (Right) problems. The RL-SANE runs used a β value of 50 for the mountain car problem and 10 for the double inverted pendulum problem.	25
4.4	This figure shows the structure of typical solution neural networks for the double inverted pendulum problem. The β value for RL-SANE was set to 10.	26
4.5	This graph shows the average performance over all initial settings of the state bound from 10 to 100 on the double pole balance problem.	29
4.6	This graph shows the best performance observed for any single state bound setting. For Fixed the best state bound setting was 10. 20 for Small and Large mutation. . .	29
4.7	These graphs show the average performance for all the methods individually as well as bars that show their maximum and minimum observed performance. Left is Fixed, middle is Small mutations, and right is Large mutation.	29
4.8	5x5 GridWorld environment	30
4.9	Individual agent performance without Kalman filter	30
4.10	Individual agent performance with Kalman filter	31
4.11	Average local reward of all agents with respect to the number of agents in the simulation. No Kalman filter	32
4.12	Average local reward of all agents using a lower learning rate. No Kalman filter . .	32
4.13	Average local reward of all agents using Kalman filter approach	33
4.14	Effect of learning rate on performance with 10 agents using Kalman filters	34
4.15	Effect of learning rate on performance with 100 agents using Kalman filters	34
4.16	Effect of epsilon (exploration rate) on performance with respect to the number of agents	35
4.17	Effect of Kalman variance on performance with 10 agents using Kalman filters . .	36
4.18	Effect of Kalman variance on performance with 100 agents using Kalman filters . .	36

4.19	This graph shows the bank balance of 5 different agents in a sample run of the ART testbed. On this run the AMS is competitive with agent UNO, the winner of the 2007 competition.	40
4.20	These graphs show the actual and perceived affinity values recorded by the AMS agent in the experiment shown in Figure 4.19. (Upper Left) Honest Agent, (Upper Right) Cheat Agent, (Lower Left) Simple Agent, and (Lower Right) UNO Agent. .	41

List of Tables

4.1	This table shows the results for experiments comparing RL-SANE to CMAC on the Mountain Car and Double Pole Balance problems. CMAC was run utilizing various parameters. Denoted with the labels CMAC $N - M$, N is the number of tiles per tiling and M is the number of tilings used. “-” signifies that the run was unable to find a solution due to either not converging to a solution or by failing from exceeding the memory capacity of the test machine.	26
-----	---	----

1 Executive Summary

This report details three efforts pursued by the agents in-house research team. The projects described in this report are: neuroevolutionary state abstraction for reinforcement learning; credit assignment for multi-agent reinforcement learning; and self-modeling for improved distributed reputation and trust management. Background information describing the problem domains and relevant technologies is provided. The approaches are described in detail and experimental results supporting the approaches are also provided.

Reinforcement learning technologies have the potential to solve many of our military's decision problems in a robust and effective manner. However, reinforcement learning cannot be applied to most real world problems due to their large state spaces. State abstraction methods reduce the size of a problem's state space to make them more approachable. In this report we describe a novel approach to state abstraction called Reinforcement Learning using State Aggregation via Neuroevolution (RL-SANE). RL-SANE automates the process of state abstraction and scales towards problems with large state spaces better than previous approaches. An experimental analysis of RL-SANE comparing it to other state abstraction methods is given. Also provided are previously unpublished results that demonstrate an improvement made to the RL-SANE approach.

Single agent reinforcement learning assumes that any changes made to the world state is the result of actions taken by the agent. In multi-agent domains this is not a reasonable assumption and a mechanism for determining which agents were responsible for changes to the world state are necessary for effective learning. This problem is known as the structural credit assignment problem and it is described in great detail with the report. We describe a distributed approach to this problem that uses a Kalman filter to attempt to derive an appropriate credit assignment for each agent making previously unlearnable problems learnable. Also provided are experiment results that demonstrate the effectiveness of this approach in a simple domain and a discussion on how the approach may be improved through inter-agent communication.

Autonomous agents operating in domains with other agents that may have conflicting goals need an appropriate means for determining reliable, reciprocal, and trust worthy partners. Trust and reputation management techniques attempt to learn through interactions with other agents in the system which ones are and are not to be trusted. In this report we describe our approach, the Affinity Management System (AMS), that attempts to estimate the trust other agents have for it as well as how much to trust other agents. By modeling the trust other agents should have for an AMS agent it is able to maintain positive collaborative relationships with trustworthy agents in the system. A full description of AMS as well as preliminary results for AMS in a competitive trust and reputation management domain are provided.

2 Introduction

Software agents are programs that have the ability to perceive the environment that they operate in, make decisions based on the observed state, and take actions that have an impact on the world state. They are an enabling technology that supports rapid, automated, distributed decision making. Agent technology has the potential to improve Air Force command and control (C2) by reducing manpower requirements, reaction times, and by making decision systems more robust in handling new situations. The research presented in this report are the products of three years of agents related in-house research. We present work on state abstraction methods for reinforcement learning, credit assignment for multi-agent learning, and distributed reputation management.

The first project presented is the development of an algorithm that automates the state abstraction process to enable reinforcement learning in domains with large state spaces. State abstraction is vital to machine learning technologies because they have difficulty in domains with large state spaces. In practice the process of state abstraction has been manual requiring extensive domain knowledge and trial and error. Reinforcement Learning using State Aggregation via NeuroEvolution (RL-SANE) is an algorithm developed in-house that finds effective state abstractions with minimal user input. It is a novel combination of neuroevolution and traditional reinforcement learning technologies. This report provides a detailed description of the algorithm and experiments that show RL-SANE's ability to abstract state spaces allow it to learn and solve difficult control problems faster than competing algorithms.

Assigning proper credit to members of a multi-agent team for global rewards is the subject of the second project. The previously described project was concerned with single agent learning and makes an assumption that only the agent's actions can change the world state. In scenarios where multiple agents are operating and learning independently this assumption does not hold true. Changes to the world state is a product of the actions taken by all agents operating in the environment. Factoring out the contribution of a particular agent from a gross reward signal will enable the agent to successfully learn in these domains using traditional reinforcement learning. However, the process of decomposing global rewards into local rewards is an open research problem. In this report we describe our investigation into this structural credit assignment problem, discuss one promising approach that utilizes a Kalman filter to derive local rewards, and describe an extension to the approach that we believe will improve its learning performance.

Rapid decision making is a byproduct of an agent's autonomy. Agents do not necessarily have to wait for user input before making a decision, allowing them to make decisions as quickly as a machine can process the agent's algorithms. In environments where autonomous agents interact with other agents that have potentially conflicting goals, mechanisms for learning which other agents

to cooperate with without user intervention are required. These mechanisms are known as trust and reputation management schemes and they are designed to prevent other agents from taking advantage of the agent employing the scheme. In this report we describe an approach called the Affinity Management System (AMS) that uses *perceived* trust to improve the agents trust and reputation management. We provide a description of the approach and preliminary results of its use in a competitive multi-agent domain designed to exercise trust and reputation management approaches.

In the following chapter, 3, we give background material necessary for an understanding of each project and then describe our approaches in detail. Chapter 4 provides the results and analysis of experiments run in support of each project. The report concludes in chapter 5 with a discussion of the findings and directions for future research.

3 Methods

3.1 Motivation

3.1.1 Reinforcement Learning for Rapid Decision Making in Real-Time Environments

RL-SANE and our other work in reinforcement learning came about from the desire to develop an agent capable of learning how to play Asynchronous Chess [12]. Asynchronous Chess, or AChess, is an agent technology evaluation platform developed in-house. It is a competitive game that two or more agents can play that is similar to chess except that players are able to move as many pieces as they want whenever they want. The only restriction placed on the players is how often pieces may be moved. This change transforms the familiar game of chess into a challenging real-time strategy game that can be used to evaluate new technologies for rapid decision making. Because the game is similar to chess it is straight forward to develop heuristically based agents that perform well in the environment. However, the game is different enough from standard chess that much of the known tactics and strategies that work for that domain do not transfer to AChess requiring novel strategies. During the development of several heuristic based agents it became apparent that a learning approach would be ideal for adapting to the strategies of various opponents and discovering novel strategies. Also, a learning approach would be of high value because the technology developed can be transferred to many other domains outside of AChess.

3.1.2 Coordinated Multi-Agent Learning

As computational hardware becomes smaller and more ubiquitous there is an increasing need for algorithms that can be effectively distributed across a collective of independent platforms. These individual elements, often referred to as agents in the machine learning community, should exhibit both local autonomy as well as contribute to a global goal structure that benefits the system as a whole.

Many joint task environments provide a reward that is based on the performance of the collective, making it difficult to assign reward accurately to individual agents based on their performance. While it's possible to design an environment with component rewards the task becomes difficult when there are a large number of agents or complicated tasks for which it is not clear what the pareto-optimal solution is. If we hope to apply distributed learning methods to a wider range

of problems it is necessary to find methods for determining reward distributions among multiple independent learners.

We focus on cooperative, model-free multi-agent environments that provide a team-based global utility based on the performance of all agents. We use the term model-free in the sense that the only information an individual agent has to make a decision is the state of the world and the global reward(s) provided by the environment. While additional knowledge could improve performance we first seek to find a domain independent solution that does not require a priori knowledge.

3.1.3 Distributed Reputation Management

Agents operating in persistent open environments with other agents require an ability to determine whether or not to cooperate with the other agents. Environments are open when there is little to no control over what entities are allowed to participate in the system. These environments are appealing in that they enable open collaboration and can scale very easily because they do not require centralized bookkeeping. However, without control there is no protection provided to guard the system against agents with malicious intent. In particular, we are referring to agents that have the objective of burdening the system by consuming resources, such as computation power and time, without providing a benefit. This type of agent is known as a free-loader. Trust and reputation management systems provide agents with a method of protecting themselves in open environments while discouraging malicious behavior of others by propagating negative feedback about potential free-loaders in the system. Beyond free-loading agents that do not intend on adding benefit to a system, we must also be aware of corruptive agents that are attempting to inject false information (liars) into a system. Identifying both free-loading and corruptive behavior will lead to increased efficiency of the overall system.

The interaction of the Air Force's information systems with those of coalition partners and non-governmental agencies continues to increase and become more open. The potential for malicious activity increases as these systems become more autonomous. Research into trust and reputation systems will provide us with a means to protect our information systems from inside and outside influences that attempt to disrupt their operations.

3.2 Background

This section provides background information necessary to understand the approaches taken by the three projects discussed in this report. It begins with a discussion of reinforcement learning and state abstraction followed by a description of the algorithm RL-SANE. Following that is a discussion of the multi-agent credit assignment problem. The section concludes with a discussion of distributed multi-agent reputation management.

3.2.1 Reinforcement Learning and State Abstraction

Reinforcement learning (RL) is a form of unsupervised machine learning that attempts to derive optimal policies for problems through positive and/or negative feedback it receives from the environment it is operating in [41]. This form of machine learning is particularly interesting in that, unlike supervised learning approaches, RL methods do not require an oracle or extensive domain knowledge to guide their learning. Because of this feature, RL methods are able to derive solutions to problems that humans do not know how to solve and find better solutions to problems humans can solve.

RL algorithms are applied to problems that can be expressed as a Markov Decision Process (MDP) [5]. Formally, a MDP can be described as a tuple $\langle S, A, T, R \rangle$ where:

- S is a finite set of states of the world. States are defined by the features of the environment. Examples of features include aircraft position, velocity, health, weapon status, target status etc.
- A is a finite set of actions.
- $T : S \times A \rightarrow S$ is the state transition function, giving for each world state and action a probability distribution over world states. We can write this as $T(s, a, s')$ for the probability of reaching state s' given that we are in state s and take action a .
- $R : S \times A \rightarrow R$ is the reward function, giving the expected immediate reward gained by taking each action in each state. We can write this as $R(s, a)$ for the expected reward for taking action a in state s .

RL algorithms attempt to learn a policy, π , to solve the problem. A policy is a mapping from the state space S to the action space A that prescribes the appropriate action to take given any state. Given a state s , where $s \in S$, the policy will dictate the next action for the learner to take, $\pi(s) \rightarrow a$. The objective of the RL algorithm is to derive the optimal policy, π^* , that maximizes the aggregate reward that the learner/agent receives while operating in the environment. RL algorithms are able to learn π^* by estimating the value of observed states based on the rewards received from visiting those states. Formally the value of a state, $V(s)$, given a policy, π , can be defined as:

$$V^\pi(s) = \sum_{a \in A_s} \pi(s, a) [R(s, a) + \gamma \sum_{s'} T(s, a, s') V_\pi(s')]$$

γ is a discount factor used to adjust the relative importance of near term or long term rewards. Given enough samples RL algorithms are able to correctly learn the value of the states of the problem and the policy π^* that enable the agent to reach the most valuable states. Popular algorithms designed and proven to learn π^* in this way include Q -learning [45] and Sarsa [41].

Current RL approaches do not scale well as the size of a problem's state space grows. RL algorithms have to experience the effect of trying actions in every state s a number of times in order to learn accurate estimates of $V(s)$ and converge upon π^* . Koenig and Simmons determined that the complexity of RL algorithms is NP-hard with respect to the size of the state space [20]. In

other words, as the number of states an RL agent experiences increases the amount of computation required to successfully derive a solution increases exponentially. This is a significant issue that has limited the applicability of RL algorithms to relatively simple problems and it is further compounded by what is called the “curse of dimensionality” [6]. The phenomenon that the curse refers to is that the dimensionality of a state space increases every time a new feature is added to describe a problems state. So, as the number of features used to describe the state increases the size of the state space grows exponentially.

Although learning over a raw state space can be prohibitively expensive, in most domains much of the state space is irrelevant or redundant with respect to solving the problem. There are a number of methods that can be used to reduce the size of a state space to a manageable size. These methods include feature set reduction [47], state abstraction, and heirarchical task decomposition [5]. In this report we focus on state abstraction.

State abstraction is the process of classifying large areas of *ground* state space as singular abstract states. The *ground* state space refers to a literal interpretation of the state where every combination of feature values describes a unique state [21]. The use of this interpretation of the state space is often not necessary. In most domains similar combinations of feature values can be generalized as being the same state with little or no impact on the RL algorithm’s ability to solve the problem. Learning over an abstract state space greatly increases the speed at which RL algorithms are able to learn problems. Because the RL algorithm is reasoning over an abstract state space that generalizes over the *ground* state space it does not have to experience every *ground* state to learn a policy for those states.

Methods for performing state abstraction include tile coding [40], radial basis functions [41], and neural networks [43]. These algorithms are examples of fixed state abstraction approaches. They require a considerable amount design considerations when being applied, and because they are fixed they do not adapt well to changes to the problem. In section 3.3.1 we describe RL-SANE which is a new approach that automates the state abstraction process and improves learning performance over traditional fixed approaches.

3.2.2 Multi-Agent Reinforcement Learning

In environments where the reward signal is a single global utility, we must ask, “how do we distribute credit to each action in the current episode which lead to the reward signal it received” This challenge, known as the temporal credit assignment problem, lies in the proper distribution of reward to the actions that contributed to the solution. In single agent learning temporal credit assignment is used to recognize how much individual actions, as part of a series of actions, contribute to local and global rewards. This interest has led to the development of a family of reinforcement learning algorithms called temporal differencing methods [41] that have proven to be very effective.

Multi-agent systems introduce a new set of challenges to the unsupervised learning process that are not present in single agent environments. Therefore, some method is needed to assign the proper amount of credit to each of the agents in a collective in an effort to maximize global utility. This is especially difficult because many environments that involve multiple agents return a single

global reward and not individualized rewards for each agent. An attempt to unify temporal and structural assignment has been proposed in [1] but relies on the assumption that the actions taken by individual agents are not concurrent and can be ordered serially.

Scalability still remains a significant challenge in many types of multi-agent systems. [27] This is especially true when the global utility contains contribution (or lack of contributions) from all agents in the system. In addition to computational scalability issues, multi-agent RL with a global reward signal suffers from interference in the form of changing rewards that are influenced by the performance of all agents in the system. Each agent is using the global reward to judge its own performance but the performance of all agents contributes to that signal. In a single agent MDP there is an assumption that the state and reward provided by the environment are a direct result of the action taken by the agent. Because other agents contribute to the reward and in some cases affect the state of other agents this assumption no longer holds.

The COIN (COLlective INTelligence) framework, which is summarized in [44], describes a set of relations between world utility and local utility that we use to characterize agent interactions with both the environment and each other. This relationship is defined by two properties referred to as factoredness and learnability.

The factoredness property represents the influence that an agent has on the reward, meaning that when the local reward increases as should the global reward. Formally, if all other local rewards are fixed an increase in local reward for one agent should never result in a decrease in global reward. This property insures that individual agents can use the global utility as an indication of their local performance even if it's obscured by the local rewards of the other agents in the system.

The learnability property indicates to what degree the global reward is sensitive to an individual agent's choices as opposed to determined by the other agents in the system. This represents the ratio of signal to noise that each agent must deal with. As the number of agents in the system increases the learnability decreases. In severe cases the individual contribution to the reward could be so small that it's indiscernible from minor noise. In [35] two agents use RL to coordinate the pushing of a block without any knowledge of each other's existence. While this is similar in the sense of being model-free it only involves two agents, which provides for high learnability.

3.2.2.1 Related Work

Reinforcement learning has proven to be an effective learning strategy in single agent environments [41] but until recently limited attention has been given to applying reinforcement learning to multi-agent environments. [36] Some methods for estimating local utility have been proposed, but in many cases these methods rely on a model of the agents, the environment, or some form of communication. [32] uses simulated power grids to explore different methods for distributed RL based on both global and local rewards but relies on there being local reward feedback, communication between nodes, and a known structure of the world that included neighboring nodes. [16] uses a coordination graph between agents to maximize subsets of the joint Q-learning process via pairwise relations. While this does not rely on a known structure of the environment it does require knowledge of the existence of other agents in the system.

Holland Bucket Brigade [18] is a well-known algorithm for assigning credit to individual clas-

sifiers in a collection using iterative training based on the joint accuracy. While this solves a temporal credit assignment for the structural distribution of credit, MCAP differs in that it is solving multiple temporal credit assignment (one for each agent) that are provided rewards by a single structural credit assignment. Because the latter approach involves more interactions between individual learning components it is significantly more complex.

We were unable to find any approaches to this problem that operated without some knowledge of the other agents in the system. While many environments provide information about the agents involved in finding the solution it is likely that some amount of limited perception will exist, motivating solutions that require as little knowledge as necessary to operate effectively.

3.2.3 Multi-Agent Trust and Reputation

In persistent environments, such as peer-to-peer (P2P) systems, where cooperation amongst agents is necessary, trust and reputation are two valuable concepts that help agents intelligently select cooperative partners. Effective agents are able to determine which agents can be reliably called upon for support and which agents should be avoided for being unreliable or uncooperative. Without a proper trust and reputation management system, agents operating in an open environment run a serious risk of being exploited by selfish opportunistic agents. Because of the problem's importance, significant attention has been paid to it over the past 20 years from experts in the field of AI, game theory, economics, and cognitive science[31].

3.2.3.1 Defining Trust and Reputation

Trust, as defined by Dasgupta[10], is a belief an agent has that the other party will do what it says it will and will reciprocate, even given an opportunity to defect to get higher payoffs. It is based on beliefs derived from direct interactions with other agents and via an agent's reputation. Reputation in multi-agent systems is a measure of an agent's standing or trustworthiness in an agent society. It is shared by agents in the form of opinions that are spread either through diffusion or retrieved from a centralized repository.

Trust can be derived from a combination of both knowledge of an agent from direct interactions or its reputation. Deriving trust from direct interactions is generally more desirable because opinions from other agents can be false or be semantically hard to evaluate. However, in environments where direct interactions are few and far between or potentially catastrophic, the use of reputation may be necessary. Reputation values are generally less expensive to obtain and more abundant. Mechanisms for sharing and using reputation reliably is the subject of many recent works [30, 34, 8, 24].

3.2.3.2 Levels of Trust

In open environments where agents can have different motivations or goals there are often incentives for lying or not participating in reciprocal behavior. Detection and avoidance of deceitful and/or "free-loading" agents are the goals of trust and reputation management systems. Agents can use computational trust models to identify good partners, protect themselves from bad ones,

and even influence other agents’ behaviors. The agents in the system which are found to be unreliable or selfish will be avoided by the other agents removing the incentive to lie or free-load. This phenomenon is termed as the *shadow of the future* and was shown in Robert Axelrod’s seminal work “Evolution of Cooperation”[4] where he showed that a cooperative relationship between two agents can be evolved in the Prisoner’s Dilemma game via a “tit for tat” strategy. *Shadow of the future* refers to the expectation of reciprocity or retaliation in future interactions that encourages cooperative behavior. By ousting liars and selfish agents the agents using a trust and reputation mechanism shape a society that removes the incentives for those behaviors and encourages the society to be more cooperative as a whole.

There are two different forms of trust in multi-agent systems that past research has focused on: system-level trust and individual-level trust[28]. System-level trust refers to trust metrics that are enforced and maintained by a centralized system or the interaction protocol itself. Examples of this type of trust include centralized rating systems, like those used by eBay and Amazon[28], and protocols such as Vickrey auctions[28]. The goal of this type of trust management system is to provide agents with some level assurance that interactions are reliable and safe. These methods try to limit the individual agent’s responsibility to find partners. This area of research is very important to designers of multi-agent systems, but is out of the focus of the research discussed in this report.

Individual-level trust refers to trust metrics built upon direct interactions with other agents and through the diffusion of third party opinions[28]. Research in this area has agents attempt to infer their own model of trust for other agents directly or through communication with others. These types of trust management schemes are desirable in that they are fully distributed and will work in unstructured open environments. Focus in these works is placed on giving the individual agent a means for selecting and keeping track of which other agents are equitable collaboration partners and which are not. It includes works that seek to determine trust values either through direct interaction only [22, 33, 26] and through the diffusion of reputation [30, 34, 29]. There are a number of open challenges associated with maintaining individual trust such as: how best to infer trust from experiences, determining defection or levels of defection from observations, modeling of opponent intentions, proper identification of agents, and the semantic exchange and interpretation of reputation.

3.3 Approaches

3.3.1 RL-SANE

Reinforcement Learning using State Aggregation via NeuroEvolution (RL-SANE) is an algorithm developed in-house to automate the process of state abstraction. It is a unique combination of neuroevolution and reinforcement learning technologies. Neuroevolution (NE) is an area of research that trains and/or constructs artificial neural networks through an evolutionary process such as a genetic algorithm. Populations of neural networks are initialized and individually evaluated on a problem that provides a fitness measure. The networks that are the “most fit” will survive to compete in the subsequent generation, whereas the “less fit” networks will be replaced with new random networks or mutated variations of the more fit networks. This evolutionary process will

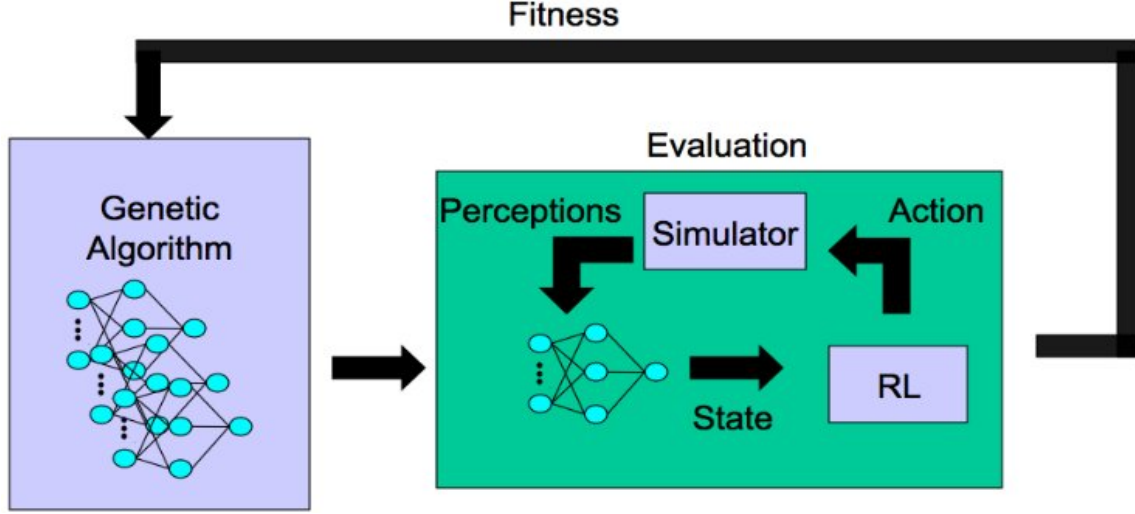


Figure 3.1: A graphical depiction of the RL-SANE algorithm

create networks that, over the course of many generations, maximize their fitness and hopefully solve the problem at hand. NE was chosen as a way for providing customized neural networks for the purpose of aggregating states. Neural networks (NN) are function approximators and they have been successful in abstracting state spaces in the past [43].

RL-SANE uses NE to evolve NNs that are bred specifically to interpret and aggregate state information which abstracts the state space for the RL component. The NNs take the raw perception values and filter the information for the RL algorithm by aggregating inputs together that are similar with respect to solving the problem¹. This reduces the state space representation and enables traditional RL algorithms, such as Sarsa(λ), to function on problems where the raw state space of the problem is too large to explore.

The NNs utilized by RL-SANE map the raw multi-dimensional perceptual information onto a single dimensional continuum that is used to represent the abstract state space. The continuum is a real number line in the range $[0..1]$. A bound on the size of the abstract state space, β , is used to divide the continuum into a number of discrete areas. All points on the continuum within a single area are considered to be the same state by RL-SANE and are labeled with a unique state identifier. The neural networks are given the raw perceptions as input and the output provided is a state identifier, this is the state aggregation process. RL-SANE uses this state identifier as input to the RL component as an index into its Q -table. The Q -table is used to keep track of values associated with taking different actions in specific states. The networks which are found to be the best at grouping perception values that are similar with respect to solving the problem will enable the RL component to learn how to solve the problem more effectively. See figure 3.1 for graphical depiction of how the algorithm functions and Algorithm 1 for a pseudo code description of the algorithm.

¹“With respect to the problem” in this paper refers to groupings made by NE that improve RL’s ability to solve the problem, not necessarily the grouping of similar perceptual values.

Algorithm 1 RL-SANE($S, A, p, m_n, m_l, m_r, g, e, \alpha, \beta, \gamma, \lambda, \epsilon$)

```
1: //S: set of all states, A: set of all actions
2: //p: population size,  $m_n$ : set of all states,  $m_l$ : link mutation rate
3: // $m_r$ : link removal mutation rate, g: number of generations
4: //e: number of episodes per generation,  $\alpha$ : learning rate
5: // $\beta$ : state bound,  $\gamma$ : discount factor,  $\lambda$ : eligibility decay rate
6: // $\epsilon$ :  $\epsilon$ -Greedy probability
7:  $P[] \leftarrow \text{INIT-POPULATION}(S, p)$  // create a new population P of random networks
8:  $Q_t[] \leftarrow$  new array size p // initialize array of Q tables
9: for  $i \leftarrow 1$  to g do
10:   for  $j \leftarrow 1$  to p do
11:      $N, Q[] \leftarrow P[j], Q_t[j]$  // select network and Q table
12:     if  $Q[] = \text{null}$  then
13:        $Q[] \leftarrow$  new array size  $\beta$  // create a new Q table
14:     end if
15:     for  $k \leftarrow 1$  to e do
16:        $s, s' \leftarrow \text{null}, \text{INIT-STATE}(S)$  // initialize the state
17:       repeat
18:          $s'_{id} \leftarrow \text{INT-VAL}(\text{EVAL-NET}(N, s') * \beta)$  // determine the state id
19:         with-prob( $\epsilon$ )  $a' \leftarrow \text{RANDOM}(A)$  //  $\epsilon$ -Greedy action selection
20:         else  $a' \leftarrow \text{argmax}_l Q[s'_{id}, l]$ 
21:         if  $s \neq \text{null}$  then
22:            $\text{SARSA}(r, a, a', \lambda, \gamma, \alpha, Q[], s_{id}, s'_{id})$  // update Q table
23:         end if
24:          $s_{id}, s, a \leftarrow s'_{id}, s', a'$ 
25:          $r, s' \leftarrow \text{TAKE-ACTION}(a')$ 
26:          $N.\text{fitness} \leftarrow N.\text{fitness} + r$  // update the fitness of the network
27:       until  $\text{TERMINAL-STATE}(s)$ 
28:     end for
29:      $Q_t[j] \leftarrow Q[]$  // update array of Q tables
30:   end for
31:    $P', Q'_t \leftarrow$  new arrays size p // array for next generation
32:   for  $j \leftarrow 1$  to p do
33:      $P'[j], Q'_t[j] \leftarrow \text{BREED-NET}(P[], Q_t[])$  // make a new network and Q table based on parents
34:     with-probability  $m_n$ :  $\text{ADD-NODE-MUTATION}(P'[j])$ 
35:     with-probability  $m_l$ :  $\text{ADD-LINK-MUTATION}(P'[j])$ 
36:     with-probability  $m_r$ :  $\text{REMOVE-LINK-MUTATION}(P'[j])$ 
37:   end for
38:    $P[], Q_t[] \leftarrow P', Q'_t[]$ 
39: end for
```

The RL component provides a fitness metric for each network to the NE component by reporting the aggregate reward it received in attempting to learn the problem. This symbiotic relationship provides the NE component with a fitness landscape for discovering better networks. NE also provides the RL component with a mechanism for coping with complex state spaces, via abstraction. Reducing the complexity of the state space makes the problem easier to solve by reducing the amount of exploration needed to calculate a good policy.

3.3.1.1 Related Work

What differentiates RL-SANE from other algorithms that use NE to do learning, such as in NEAT, is that RL-SANE separates the two mechanisms that perform the state aggregation and policy iteration. Previous NE algorithms[37][46] use neural networks to learn a function that not only encompasses a solution to the state aggregation problem, but the policy iteration problem as well. We believe this overburdens the networks and increases the likelihood of “interference”[9]. In this

context, “interference” is the problem of damaging one aspect of the solution while attempting to fix another. RL-SANE, instead, only uses the neural networks to perform the state aggregation and uses RL to perform the policy iteration which reduces the potential impact of interference.

3.3.1.2 Bounding of the Abstract State Space

Selection of an appropriate or optimal state bound (β) is of great importance to the performance of the algorithm. If the selected state bound is too small, sets of perception values that are not alike with respect to the problem will be aggregated into the same state. This may hide relevant information from the RL algorithm and prevent it from finding the optimal, or even a good, policy. If the state bound is too large, then perception sets that should be grouped together may map to different states. This forces the RL component to experience and learn more states than are necessary. Effectively, this slows the rate at which the RL algorithm is able to learn a good policy because it is experiencing and exploring actions for redundant states, causing bloat. Bloat makes the Q -tables larger than necessary, increasing the memory footprint of the algorithm. In [49] we performed an analysis of the impact of the β parameter on the performance of the algorithm for our benchmark programs. While we found that improper settings did have major adverse affects on the performance, the effects were consistent and could provide a landscape for a hill climbing algorithm to learn the proper setting. Below we describe the an approach we used to attempt to automate the parameter selection process.

3.3.1.3 State Bound Mutation

The RL-SANE algorithm accepts one parameter, the state bound, which controls the number of bins in the abstracted state space. A good estimation of this parameter will enable the neural network to abstract the state observations well and quickly converge on an optimal policy. To this end, instead of requiring the user to set a fixed state bound prior to learning, we take this state bound as a suggestion and allow the neural network to mutate it as learning goes on in an effort to find a better state bound than the user can provide. We tested two variations on this idea, the first allows the neural network to mutate the state bound one bin at a time, while the second allows for multiple bins to be added or removed per mutation.

The RL-SANE algorithm using the single bin mutation works as follows. An additional chromosome is added to each of the neural networks, representing the current state bound. During the mutation phase, this chromosome can be randomly selected and the value of the state bound is either incremented or decremented. In either case, the current values for each action in each bin need to be redistributed according to the new number of bins. For example, if the state continuum is currently divided into four bins and the state bound is incremented, the action values for each of the four current states must be redistributed across the five states. For new bins that fall completely within an old region, the new action values are a fraction of the old values; this fraction is the percentage of the old bin that is overlapped by the new bin. For bins that straddle two old ones, the new values are the sum of the corresponding fractions of the two bins that it overlaps. Computing the new values in this way allows all of the previously learned values to be reused in future learning.

The multiple bin mutation RL-SANE generalizes this idea. Since the initial state bound estimate may be far away from a good state bound value, moving in single steps towards this value may not be sufficient. Instead of limiting the neural network to only incrementing or decrementing the state bound, we allow it to select a random number of bins to add or remove, and the user can specify the maximum number of bins that can be added or removed in a single step. This allows the neural network to converge to a good state bound value faster than with the single step approach. Once again, the values for each action must be redistributed across the new bins when the state bound changes. There is the possibility that new bins might overlap more than two old ones, in which case each new bin sums up all of the values from the old bins that it replaces. For old bins that are only fractionally overlapped, only the corresponding fraction of the old values contribute to the new values.

3.3.2 Kalman Filter for Structural Credit Assignment

The Kalman Filter (KF) [19] is an optimal (unbiased) estimator for problems with linear Gaussian characteristics. A recursive filter, the KF estimates the true state of a system based on numerous observations of the state. The algorithm is quite simple, producing estimates by performing two stages, prediction and update. In the prediction phase, the state and covariance estimates from the previous iteration are projected forward to the current observation period using state transition and process noise covariance matrices. Next, the update stage, corrects the predicted state and covariance estimates through a weighting factor known as the Kalman gain. This process is repeated iteratively over the entire observable period of the state.

A Kalman Filter was used in [17] to generate local utility estimates based on the global utility received at each time step. In essence, this approach creates a mapping from states to rewards based on the variance of the global utility with respect to the states visited. As a result each agent forms a more accurate estimation of local utility for each state through repeated visits and varying global utilities. This approach does not only function without any models of the agents or environment, it does not even need to know that the other agents in the system even exist. This makes it a powerful method of estimation usable in a wide range of tasks.

In this approach, the Kalman filter is used to generate estimations of state utilities from noisy data. The global reward at time t is a linear combination of the local reward for being in state i and a noise term b at time t :

$$g_t = r(i) + b_t \quad (3.1)$$

The original Kalman Filter utilized matrices to store internal values and required matrix multiplication to perform updates at each time step. As a result, implementations of this approach require significant computation as well as knowledge of exactly how many unique states will be encountered that the data structures can be initialized. Because of this we were limited in the environments we could use and the Kalman Filter also processed information for states it may have never visited. We have developed a matrix-free implementation of the Kalman Filter that not only removes some of the unnecessary structures and operations (such as utilizing only half of the covariance matrix since the covariance of state A and B is equivalent to the covariance of states B

Algorithm 2 KALMAN LEARNER(\vec{s}, g)

```
1: //  $\vec{s}$ : state vector
2: //  $g$ : reward signal for current state
3:  $CP[] \leftarrow 0$  // Initialize covariance of current state vector
4:  $CP_{noise} \leftarrow 0$  // Initialize covariance of noise with respect to current state
5:  $s = \text{GET\_STATE\_ID}(\vec{s})$ 
6: if  $|S| == 0$  then
7:    $E[s] \leftarrow g$ 
8:    $E_{noise} \leftarrow 0$ 
9:   EXPAND_STRUCTURES()
10: end if
11: if  $S[s] == \text{null}$  then
12:    $E[s] \leftarrow g - E_{noise}$ 
13:   EXPAND_STRUCTURES()
14: end if
15:  $PG_{noise} = PG_{noise} + KV$ 
16:  $resid = g - (E[s] + E_{noise})$ 
17: for  $i \leftarrow 0$  to  $|S| - 1$  do
18:    $CP[i] \leftarrow P[s][i] + PG[i]$ 
19: end for
20:  $CP_{noise} = PG[s] + PG_{noise}$ 
21:  $val = CP[s] + CP_{noise}$ 
22: for  $i \leftarrow 0$  to  $|S| - 1$  do
23:    $K[i] = CP[i] / val$ 
24:    $E[i] = E[i] + K[i] * resid$ 
25: end for
26:  $K_{noise} = CP_{noise} / val$ 
27:  $E_{noise} = E_{noise} + K_{noise} * resid$ 
28: for  $i \leftarrow 0$  to  $|S| - 1$  do
29:   for  $j \leftarrow 0$  to  $|S| - 1$  do
30:      $P[i][j] = P[i][j] - K[i] * CP[j]$ 
31:   end for
32: end for
33: for  $i \leftarrow 0$  to  $|S| - 1$  do
34:    $PG[i] \leftarrow PG[i] - K[i] * CP_{noise}$ 
35: end for
36:  $PG_{noise} = PG_{noise} - K_{noise} * CP_{noise}$ 
37: return  $E[s]$ 
```

and A) but also allows for growing data structures as new states are encountered.

It is still unclear how initial values of the covariance matrix P affect the speed and accuracy of reward estimation. The simplest values we could come up with that yielded good results was using the identity matrix multiplied by some small scalar value. For all experiments a scalar value of 0.1 was used but its likely that any knowledge of the environment could be encoded into P to provide the Kalman Filter with some bootstrapping information.

3.3.3 Affinity Management System (AMS)

The Affinity Management System (AMS) is our in-house developed multi-agent trust management scheme designed for cooperative multi-agent environments[26]. Agents using AMS establish trust or affinity for other agents in the system through direct interactions. We use *affinity* to define the level of trust an agent has with another. The affinity values can range between -1 and 1 signifying extreme distrust and extreme trust respectfully. The AMS is theoretically based on the mechanism of *reciprocity* [22, 33, 30]; Affinity is increased after positive and reciprocal interactions

and decreased for negative or non-reciprocal interactions. Unlike other reciprocity-based trust and reputation management systems, the AMS also models the affinity that other agents should have for the agent(s) using AMS. By modeling the affinity that other agents have for an AMS agent, the AMS agent is better able to sustain collaborative relationships, seek out reliable collaboration partners, and evaluate the potential or future utility for cooperating.

Each agent, i , keeps a 2-tuple, \mathcal{F} , which holds the affinity values: the *actual affinity value* \mathcal{A} , and the *perceived affinity value* \mathcal{P} . We define these as follows:

$$\mathcal{F} = (\mathcal{A}, \mathcal{P})$$

$\mathcal{A}_{i,j}$ is the affinity value that i has about j (another agent in the system). This value contains information on how i 's requests have been honored by j . $\mathcal{P}_{i,j}$ is i 's perceived affinity value that j may have about i . This value is updated when j accepts or rejects a request from i . Notice that both \mathcal{A} and \mathcal{P} are kept and managed by i . Initially, $\mathcal{A}_{i,j} = 0$ and $\mathcal{P}_{i,j} = 0$.

3.3.3.1 Utility in AMS

In general, game theory defines utility to be the expected benefit for choosing an action, $A \in \{a, \bar{a}\}$, subtracted by the cost of the taking the action. That is,

$$\mathcal{U}(A) = \mathcal{B}(A) - \mathcal{C}(A) \quad (3.2)$$

In the AMS, we use the above utility definition to determine whether or not to accept a request for assistance or information in a MAS. The expected benefit is what the agent can expect in return for future interactions with the requesting agent. A cost function is used to discount the expected benefit, and is usually domain dependent.

3.3.3.2 Detecting and Penalizing Dishonest Agents

Agents operating in open competitive environments require a means for detecting and punishing dishonest behavior of other agents. If an agent has no mechanism for detecting an punishing dishonest behavior, other agents have no reason for not being dishonest if there are potential payoffs. Detecting dishonest behaviors can be largely domain specific. So, our method for detecting and penalizing dishonesty allows for heuristics to be used in the detection of false behaviors and integrates the penalty directly into the affinity values. Equation 3.3 describes how $\mathcal{A}_{i,j}$ and $\mathcal{P}_{i,j}$ are updated after an interaction with an individual agent is completed.

$$\mathcal{A}_{i,j} = \alpha \times \mathcal{A}_{i,j} + (1 - \alpha) \times v(t) \quad (3.3)$$

Dishonest behavior is detected through the calculation of $v(t)$ which is a heuristic function that is domain dependent. The term $v(t)$ in equation 3.3 is used to denote the value of the performed task t . We bound the values $v(t)$ can take to be any real value between -1 and 1 that correspond to complete dissatisfaction and complete satisfaction respectively. Negative values for $v(t)$ should be used when it is believed the other agent was deceitful. The range of $v(t)$ values can be used to describe varying degrees of honesty.

Combined with a proper heuristic for determining $v(t)$ values, this provides AMS agents with a means to detect and penalize deceitful and selfish agents with negative utility values. In addition to equation 3.3, an additional aging function for $\mathcal{A}_{i,j}$ and $\mathcal{P}_{i,j}$ is added to make them tend towards zero. This aging mechanism mimics the forgiveness mechanism in the “tit for tat” strategy[4] and prevent AMS from dismissing adaptive agents that can learn to cooperate.

3.3.3.3 Probabilistic Cooperation Based on Affinity and Utility

AMS determines whether or not to accept another agent’s request for cooperation based on the utility to be gained from the collaboration. In [26] utility was defined as the increase in affinity, which represents the potential for reciprocity, subtracted by the cost. Cost was not fully defined and the cost of all tasks were considered equal. In environments with heterogeneous tasks a utility function that incorporates not just reciprocity, but the degree of reciprocity is required.

In this section we describe a new utility calculation, inspired by Sen’s *Probabilistic Reciprocity Model*[33], that incorporates a measure of how much reciprocity is to be expected. The *Probabilistic Reciprocity Model* (PRM) keeps track of the balance of reciprocity between agents. If an agent i has done more for agent j in terms of the cost of collaborating then agent i has a positive balance $\mathcal{B}_{i,j}$ in dealing with agent j . The purpose of keeping track of the balance $\mathcal{B}_{i,j}$ is that agent i should be less inclined to cooperate with agent j the more positive the balance becomes. Agents use a sigmoidal function based on the value of $\mathcal{B}_{i,j}$ to determine the probability for accepting further requests from agent j .

$$Pr(i, j) = \frac{1}{1 + e^{\frac{\mathcal{B}_{i,j} - U}{\tau}}} \quad (3.4)$$

Equation 3.4 is the new formula based on PRM that we intend to use with AMS to determine whether or not it is appropriate for agent i to accept agent j ’s cooperation request. It returns a value between 0 and 1 that determines the probability for cooperating. The new function includes $\mathcal{B}_{i,j}$ balance information because it enables the AMS agent to handle environments with asymmetric size tasks. τ is a constant in the function used to scale the result. U , in the equation, represents the utility for accepting the request and is derived by the following function. The probability of the AMS accepting a cooperation request increases if either the utility is positive or if the balance of reciprocity is negative.

$$U(i, j, c) = (\Delta\mathcal{P}_{i,j} + \mathcal{A}_{i,j}) \times c - c \quad (3.5)$$

Equation 3.5 defines utility to be a function of the difference between benefit and cost. The symbol c represents the cost in performing the task and is defined by the environment. The benefit for cooperating is a function of how much agent i trusts agent j to be reciprocal, $\mathcal{A}_{i,j} \times c$, and the increased affinity j will have for i , $\Delta\mathcal{P}_{i,j} \times c$. Cost c is included in the derivation of the benefit because the level of reciprocation and increase in affinity should be relative to the size of the favor performed. Using this equation AMS agents will derive positive utility for helping agents they have higher affinity for or will receive higher affinity from. The function will also report negative

utility for assisting agents they have negative affinity for, preventing them from cooperating with those agents.

3.3.3.4 Related Work

In section 3.2.3.2 we mentioned the different types of MAS computation trust models. Here we will discuss three of the individual-trust based models in more detail.

- The first approach is one of the earliest computational trust models designed for MAS. Designed by Marsh in 1994[22] he designed a trust mechanism that attempted to determine whether or not to accept cooperative arrangements based on the current situation. Agents using this model keep track of three different types of trust. *Basic Trust*, the agent's overall disposition towards trusting everyone. *General Trust*, the amount of trust an agent has towards a particular agent regardless of the situation. And finally, *Situational Trust*, which is the amount of trust for the current situation derived from the general trust and the utility of effort.

This approach is similar to AMS in that they are both based on trust derived from direct interactions, they both encourage reciprocity, and neither approach has a reputation component. However, we believe dividing trust values into separate components is unnecessary and only complicates the model. AMS uses a single measure for affinity which is easier to calculate and should be less error prone. Marsh's method determines whether or not to cooperate with other agents based on statistical evidence and trust values derived from that evidence. AMS uses trust values and the utility for the cooperative interaction to make the same determination.

- Sen's Probabilistic Reciprocity Model (PRM)[33] is our next exemplary model. It is a simple mechanism based purely on the premise of reciprocal behavior. In this system each agent keeps track of the balance of reciprocation between it and all other agents. If the number and size of favors that agent i has done for j becomes sufficiently greater than what the other agent has reciprocated, i will become less likely to cooperate with j . It is a very simplistic model, but was shown to be very effective in excluding selfish agents from a population of cooperatively reciprocal agents[33].

PRM is not a trust management system in the traditional sense. It keeps no measure of trust and has no mechanism for detecting dishonest behaviors of other agents. It only keeps track of which agents did what, how much was done, and if there is an unfair balance in a reciprocal relationship. Other agents are penalized by the agents using PRM if they are found not to return favors. AMS keeps track of actual trust values between agents. By using trust values AMS agents can avoid non-reciprocal and dishonest agents. The idea of keeping a reciprocal balance is intriguing though and in section 3.3.3 we described how we borrowed from Sen's idea to improve the AMS model so that it can differentiate levels of cooperation in terms of the associated costs.

- REGRET by Sabater *et al.*[30] is one of the more advanced trust and reputation models. This model combines both direct trust and three different reputation components termed: *Witness reputation*, *Neighborhood reputation*, and *System reputation*. REGRET includes an *ontological dimension* which is used to combine the three subjective measures of reputation into a coherent and meaningful value which is used in determining whether or not to cooperate with other agents. The model is fairly complicated but modular, so certain components can be turned on or off to make a comparison to other methods, like AMS, easier and more meaningful.

This approach is very complex in comparison to the AMS model. It keeps databases of different types of interactions between agents. These databases are used to evaluate levels of trust over various contexts. This model may be more robust than AMS, but it also may not be able to scale well because it retains a significant amount of information for each interaction.

Like AMS, all of these models operate based on trust, reciprocity, or both. Aside from the differences described above, the most notable difference between these three models and the AMS model is that these models only model the trust an agent should have for others. AMS based agents also model the level of trust other agents should have for it based on past interactions. We believe AMS has an advantage over these other approaches because it can use this information to select partners that are the more likely to cooperate and maintain productive relationships between agents.

4 Results and Analysis

4.1 RL-SANE Experiments

This section provides an experimental analysis of the RL-SANE algorithm designed to show the benefit its state abstraction process provides. RL-SANE is compared against two competing and established approaches on two well known benchmark problems. The results will show that the RL-SANE method of learning with state abstraction is superior to the other two approaches.

4.1.1 Benchmark Algorithms

4.1.1.1 NeuroEvolution of Augmenting Topologies

NeuroEvolution of Augmenting Topologies (NEAT) is a neuroevolutionary algorithm and framework that uses a genetic algorithm to evolve populations of neural networks to solve reinforcement learning problems[38]. Neuroevolution is a category of research that incorporates all technologies that train or construct artificial neural networks through an evolutionary process such as a genetic algorithm. Populations of neural networks are initialized and individually evaluated on a problem that provides a fitness measure. The networks that are the “most fit” will survive to compete in the subsequent generation, whereas the “less fit” networks will be replaced with new random networks or mutated variations of the more fit networks. This evolutionary process will create networks that, over the course of many generations, maximize their fitness and hopefully solve the problem at hand.

NEAT is an example of a TWEANN (Topology & Weight Evolving Artificial Neural Network) algorithm in that it evolves both the network topology and weights of the connections between the nodes in the network[37]. The authors, Stanley et al., found that by evolving both the network topology and the connection weights, they solved typical RL benchmark problems several times faster than competing RL algorithms[37] with significantly less system resources. The algorithm starts with a population of simple perceptron networks and gradually, through the use of a genetic algorithm (GA), builds more complex networks with the proper structure to solve a problem. It is a bottom-up approach which takes advantage of incremental improvements to solve the problem. The results are networks that are automatically generated, not overly complicated in terms of structure, and custom tuned for the problem.

Two of the most powerful aspects of NEAT which allow for such good benchmark performance and proper network structure are *complexification*[39] and *speciation*[38]. *Complexification*, in

this context, is an evolutionary process of constructing neural networks through genetic mutation operators. These genetic mutation operators are: modify weight, add link, and add node [39]. *Speciation* is a method for protecting newly evolved neural network structures (specie) into the general population, and prevent them from having to compete with more optimized species [39]. This is important to preserve new (possibly helpful) structures with less optimized network weights.

NEAT was chosen for our evaluation because it is a standard for TWEANN algorithms. Because it is a TWEANN algorithm it can be applied to many problems with very few modifications. The only prerequisites for applying NEAT to a problem is to know the number inputs provided, actions available, and the reward value for any given state the problem can take. Its performance should characterize the performance of other general purpose neuroevolutionary algorithms. Unlike the other algorithms used in our evaluation NEAT is not designed to perform state abstraction, although it does perform this task indirectly. Through *complexification* NEAT is able to derive ever more complex neural networks that can approximate the correct policy of an arbitrary environment. However, it is an expensive process and does not scale well as the size of the problem’s state space is increased. The inclusion of NEAT in our analysis will help demonstrate the importance of including a state abstraction component on problems with large state spaces.

4.1.1.2 Cerebellar Model Articulation Controller

The Cerebellar Model Articulation Controller (CMAC) algorithm was introduced in [23] (then called the Cerebellar Model Arithmetic Computer) as a means of providing local generalization of the state space based on how the human brain is thought to respond to stimuli [2]. This behavior allows states that are in proximity to an observed state to learn even though those states have not been observed themselves. Generalization is critical on continuous state spaces where the probability of seeing a previously learned state approaches zero; learning an optimal policy is reliant on using the values learned on similar states to cover for this lack of repetition.

CMAC can be considered a tile coding method [40], where the state space is partitioned into a set of tiles, and values that are learned from any one state in a tile are learned for all states in the tile. Partitioning the state space into many small tiles will slow learning since not many states are learned at a time, but all learned values will be very accurate in those tiles. Conversely, if the tiles are very large then action values will be distributed quickly across many states, but there is no guarantee that two states on opposite sides of a tile should share the same action values. In this case, each state may favor a different action, but only one action can be preferred per tile, possibly preventing a correct policy from being found. This tradeoff is mitigated by overlapping layers of tiles to provide both coarse and fine grain generalization. Each observed state updates one tile per layer, and each of these tiles covers a different portion of the state space. The preferred action for a state is the action that maximizes the (possibly weighted) sum of action values across all tiles that contain that state.

The CMAC algorithm has effectively learned a number of domains including the mountain car and single pole balance [40]. More recently, it has been shown to suffer from some limitations on slightly more complicated problems like the double pole balance [14]. The main difficulty in applying CMAC is choosing a suitable way to break up the state space into tiles. If this is

done inexpertly then states that do not prefer the same action can be forced to learn together if they are both confined to a single tile. This will severely slow down, if not prevent, the learning of a successful policy. A secondary concern is the memory requirements for high dimensional scenarios. The number of tiles per mapping scales exponentially in the number of input perceptions for a problem, and storing all visited tiles can quickly become unreasonable. Hashing techniques like those mentioned in [23] can be used to place limits on the memory requirements of CMAC, but they effectively cause non-local generalization of learned values in the event of a hash collision, which can negatively impact policy convergence.

Two recent methods have been proposed to overcome these difficulties. In [48] an adaptive tile coding mechanism is investigated as a way to locate areas of the state space that need more specific values to be learned. When no action is clearly preferred by a tile it is split into two tiles where the increased specificity of the new smaller tiles can lead to a better action selection. While promising, this method is not yet mature enough to be applied on general reinforcement learning problems. The use of two CMACs, one for specific areas of the state space where more discrimination is necessary, and another to provide wide area generalization is discussed in [50]. A positive result is given for the double pole balance problem in that work, however, the difficult step of identifying which areas of the state space require more discrimination must be manually specified.

CMAC was chosen for our analysis because it is the standard approach used for abstracting state spaces for reinforcement learning algorithms. Despite being a fixed linear function approximation and having storage requirements that grow dimensionally with the state space, CMAC has been successfully applied in many domains [40, 50]. Including CMAC in our analysis will help determine the limitations of fixed state abstractions on problems with ever larger state spaces.

4.1.2 Benchmark Problems and Experimental Setup

4.1.2.1 Mountain Car

The mountain car problem [7] is a well known benchmark problem for reinforcement learning algorithms. In this problem a car begins somewhere in the basin of a valley, of which it must escape. See Figure 4.1 for an illustration of the problem. Unfortunately, the car’s engine is not powerful enough to drive directly up the hill from a standing start at the bottom. To get out of the valley and reach the goal position the car must build up momentum from gravity by driving back and forth, up and down each side of the valley.

For this problem the learner has two perceptions: the position of the car, X , and the velocity of car, V . Time is discretized into time steps and the learner is allowed one of two actions, drive forward or backward, during each time step. The only reward the learner is given is a value of -1 for each time step of the simulation in which it has not reached the goal. Because the RL algorithms are seeking to maximize aggregate rewards, this negative reward gives the learner an incentive to learn a policy which will reach the goal in as few time steps as possible.

The mountain car problem is a challenging problem for RL algorithms because it has continuous inputs. The problem has an almost infinite number of states if each set of distinct perceptual values is taken to be a unique state. It is up to the learner or the designer of the learning algorithm to determine under what conditions the driver of the car should consider a different course of action.

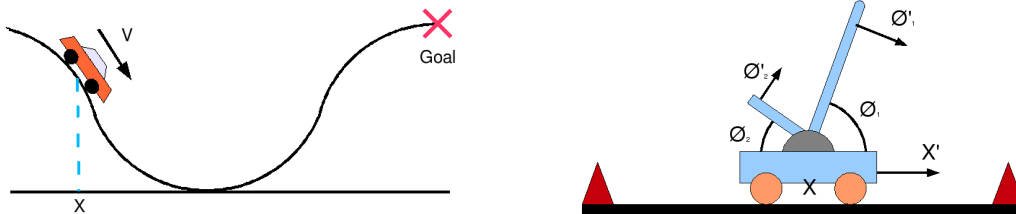


Figure 4.1: These figures illustrate the two RL benchmark problems, (Left) mountain car and (Right) double inverted pendulum balancing.

In these experiments the populations of neural networks for all of the algorithms have two input nodes, one for X and one for V , which are given the raw perceptions. NEAT networks have three output nodes (one for each direction the car can drive plus coasting) to specify the action the car should take. RL-SANE networks have a single output node which is used to identify the state of the problem.

Individual episodes are limited to 2500 time steps to ensure evaluations will end. Each algorithm was recorded for 25 different runs using a unique random seed for the GA. The same set of 25 random seeds were used in evaluating all three algorithms. Runs were composed of 100 generations in which each member of the population was evaluated over 100 episodes per generation. The population size for the GA was set to 100 for every algorithm. Each episode challenged the learner with a unique instance of the problem from a fixed set that starts with the car in a different location or having a different velocity. By varying the instances over the episodes we helped ensure the learners were solving the problem and not just a specific instance.

4.1.2.2 Double Inverted Pendulum Balancing

The double inverted pendulum balancing problem [13] is a very difficult RL benchmark problem. See figure 4.1 for an illustration of this problem. In this problem the learner is tasked with learning to balance two beams of different mass and length attached to a cart that the learner can move. The learner has to prevent the beams from falling over without moving the cart past the barriers which restrict the cart's movement. If the learner is able to prevent the beams from falling over for a specified number of time steps the problem is considered solved.

The learner is given six perceptions as input: X is the position of the cart, X' is the velocity of the cart, θ_1 and θ_2 are the angles of the beam, and θ'_1 and θ'_2 are the angular velocities of the beams. At any given time step the learner can do one of the following actions: push the cart left, push the cart right, or not push the cart at all.

Like the mountain car problem, this problem is very difficult for RL algorithms because the perception inputs are continuous. This problem is much more difficult in that it has three times as many perceptions giving it a dramatically larger state space. In these experiments each algorithm trains neural networks that have six input nodes, one for each perception. NEAT has three output nodes, one for each action. RL-SANE only has one output node for the identification of the current state.

In this set of experiments the algorithms were tasked with creating solutions that could balance

the pendulum for 10^5 time steps. Each algorithm was evaluated over 25 different runs where the random seed was modified. Again, the same set of random seeds was used in evaluating all three algorithms. Runs were composed of 200 generations with a population size of 100. In each generation, every member of the population was evaluated on 100 different instances of the inverted pendulum problem. The average number of steps the potential solution was able to balance the pendulum over the set of 100 determined its fitness.

4.1.3 Experimental Results

4.1.3.1 NEAT Comparison

Figure 4.3 show convergence graphs comparing the RL-SANE to NEAT on the benchmarks. The error bars on the graphs indicate 95% confidence intervals over the entire set of experiments to show statistical significance of the results. As can be seen from the charts the RL-SANE algorithm is able to converge to the final solution in far fewer generations and with a greater likelihood of success than NEAT in both sets of experiments. NEAT is only able to solve the entire double inverted pendulum problem set in just 2 of the 25 runs of experiments. The performance difference between RL-SANE and NEAT increases dramatically from the mountain car to double inverted pendulum problem which implies that RL-SANE may scale to more difficult problems better than NEAT.

Figures 4.2 and 4.4 show the structure of typical solution NN(s) for both RL-SANE and NEAT on both problems. In both figures the RL-SANE networks are much less complex than the NEAT networks. These results are not very surprising in that the RL-SANE networks are performing a less complicated function, state aggregation. NEAT networks need to perform their own version of state abstraction and policy iteration. The fact the RL-SANE networks do not need to be as complicated as NEAT networks explains why RL-SANE is able to perform better than NEAT on these benchmarks. If the NN's do not need to be as complex they have a greater likelihood of being produced by the GA earlier in the evolution. This result also supports our belief that RL-SANE will scale better to more complex problems than standard NEAT.

The results shown in this section clearly demonstrate that combination of NE and RL is superior to a pure NE approach on these types of control problems. This conclusion is not surprising given that the NEAT approach does not explicitly perform state abstraction, which as discussed earlier (section 3.2.1) greatly reduces the complexity of the problem. The defections of the resulting NNs in figures 4.2 and 4.4 show the extent to which the complexity is reduced and the performance of RL-SANE over NEAT provides further evidence. In addition, RL-SANE has a second advantage over NEAT in that it employs a traditional temporal differencing RL algorithm to assist it in overcoming the temporal credit assignment problem associated with problems of this type.

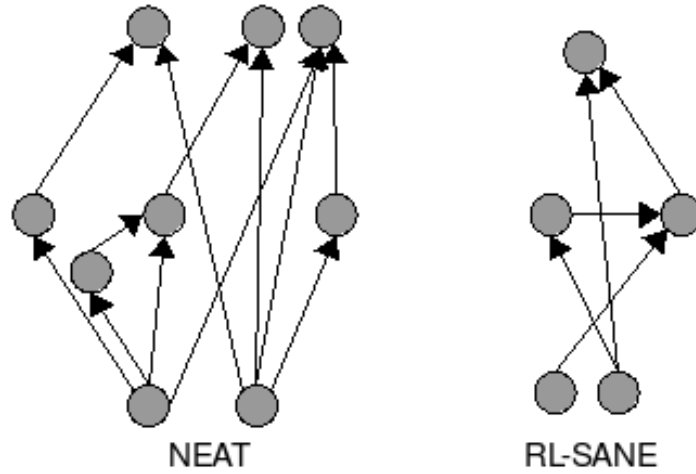


Figure 4.2: This figure shows the structure of typical solution neural networks for the mountain car problem. The β value for RL-SANE was set to 50.

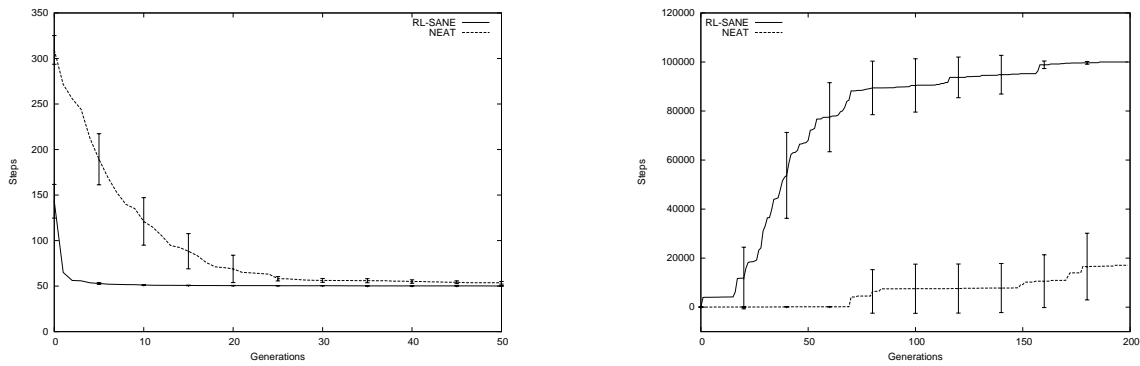


Figure 4.3: Shows the performance of the RL-SANE algorithm compared to that of NEAT on the mountain car (Left) and the double inverted pendulum (Right) problems. The RL-SANE runs used a β value of 50 for the mountain car problem and 10 for the double inverted pendulum problem.

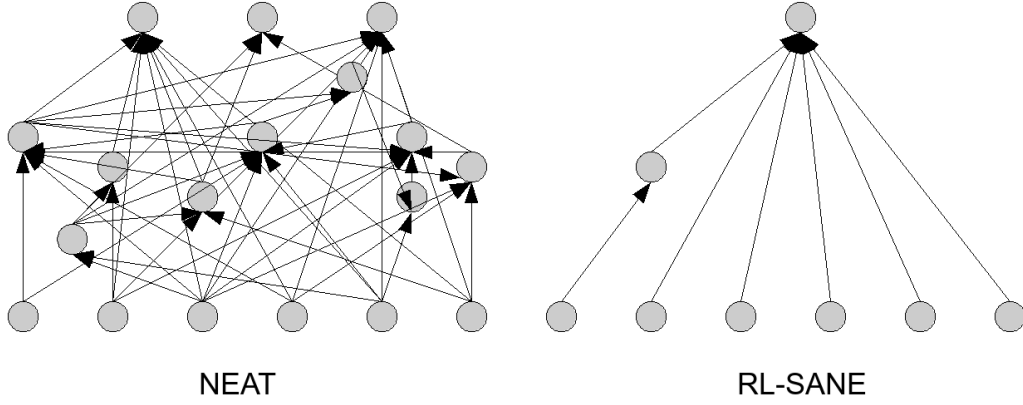


Figure 4.4: This figure shows the structure of typical solution neural networks for the double inverted pendulum problem. The β value for RL-SANE was set to 10.

Algorithm	Mountain Car			Double Pole Balance		
	Updates	States	CPU Time	Updates	States	CPU Time
RL-SANE	6.28E+07	21.64	21.42	1.80E+09	29.96	1773.46
CMAC 5-5	214825	114	0.5	-	-	-
CMAC 5-10	1866269	231	3.44	-	-	-
CMAC 10-5	164069	450	0.44	-	-	-
CMAC 10-10	153366	912	0.54	-	-	-
CMAC 10-20	145472	1831	0.76	-	-	-
CMAC 20-10	453413	3553	1.18	-	-	-
CMAC 20-20	447335	7121	2.04	-	-	-

Table 4.1: This table shows the results for experiments comparing RL-SANE to CMAC on the Mountain Car and Double Pole Balance problems. CMAC was run utilizing various parameters. Denoted with the labels CMAC $N - M$, N is the number of tiles per tiling and M is the number of tilings used. “-” signifies that the run was unable to find a solution due to either not converging to a solution or by failing from exceeding the memory capacity of the test machine.

4.1.3.2 CMAC Comparison

Here we present the results of the experiments comparing RL-SANE to CMAC. CMAC like RL-SANE, unlike NEAT, utilizes a state abstraction component to reduce the complexity of a problem’s state space and a RL algorithm to derive policies. Table 4.1 shows the results of the experiments on the mountain car and double pole balance benchmark problems. CMAC has two parameters, number of tiles per tiling and number of tilings, that are used to adjust the size and generality of the abstract state space. We ran multiple experiments varying the numbers of tiles per tiling and number of tilings from 5 to 20. It is important to remember with CMAC that the dimensionality of the abstract state space increases with the dimensionality of the ground state space, so values exceeding 20 for the parameters were not tested as memory constraints were easily exceeded. RL-SANE was evaluated using the average found over 25 runs in each domain, whereas CMAC is a deterministic algorithm and only required single runs of each parameter setting. RL-SANE in this experiment also uses the state bound large mutation operator to dynamically adjust the state space. This is the reason why the “States” metric is not an integer for the RL-SANE results. The metrics used are: Updates which shows the number of times the Q -tables were updated (less is better), States that specifies the size of the abstract state space (less is better), and CPU time which accounts for the run time required by the algorithms to solve the problems (less is better).

The results show that for the mountain car problem CMAC is able to solve the problem in much less time and requires far fewer updates than RL-SANE. RL-SANE however is able to solve the mountain car problem using fewer states which may indicate that RL-SANE’s state abstraction method is more efficient than CMAC’s. CMAC is unable to solve the double pole balance problem at all whereas RL-SANE is. In our testing, utilizing these parameters, CMAC either failed to converge to a solution or failed because it exceeded the memory capacity of the testing machine. CMAC failed to converge using lower parameter values because the abstract state space is too general hiding relevant details needed to find a good policy. The double pole balance problem has a six dimensional state space, which is the reason why even for parameter settings less than 20 CMAC exceeded the memory constraints of the computer. These findings are supported by the authors of [15] who also attempted and failed to apply CMAC to the double pole balance problem. The dimensionality of the state space and the potentially non-linear relationship between features prevent CMAC from being effective on this problem. CMAC could potentially solve this problem if more effort was spent in defining the shape and density of the tiles. However, spending extra effort in engineering CMAC to work on such problems does not make sense when an approach like RL-SANE can successfully be applied with minimal customization.

4.1.3.3 State Bound Mutation Experiments

Proper state bound values are critical to the overall performance of RL-SANE. In [49] our initial results on varying state bound values suggested that optimal state bound values could potentially be derived during the operation of the algorithm. Automating the selection of the state bound parameter improves the generality and utility of this approach. Section 3.3.1.3 details our approach to adjusting the state bound parameter on-line through the use of a mutation parameter. The new method still requires an initial setting of the state bound, but that value is adjusted by the algorithm

on-line to a more suitable setting. To test this new mutation method we compare small and large mutation variants of it to the original fixed state bound variant of RL-SANE. Initial state bound values from 10 to 100 were evaluated over each method on the double pole balancing problem. In the large mutation experiments the value of the state bound parameter value was allowed to vary by ± 5 every generation.

Figure 4.6 shows the best performance obtained by any of the approaches for a single setting of the state bound. In the graph the best large and small mutation runs are able to outperform the best fixed run. While the improvement is only marginal it does show that the adaptive state bound methods are able to converge to a solution as well as the fixed approach, so there is no loss in best performance from using them. Figure 4.5 shows the average performance of all three methods across all initial settings of the state bounds. The improvement in average performance that the large and small methods share over the fixed method is significant and it shows how the adaptive methods are less sensitive to improper initial state bound settings. Figure 4.7 further emphasizes the advantage of the adaptive methods. The graphs in figure 4.7 show the average performance of the three methods individually as well as bars that show the best and worst performance observed for each method over the testing. Fixed’s performance varies widely depending on the initial setting of the statebound whereas it does not vary nearly as much for the adaptive methods. From these results we can conclude that the adaptive mutation approaches are superior to the fixed method. There is no loss in performance and performance is less sensitive to poor initial state bound settings.

The results did not show a significant difference in the performance of the small and large state bound mutation operators. It does appear that the large mutation operator slightly outperformed the small operator but not significantly. Although these results do not demonstrate a benefit to using one over the other, we do believe that the large mutation operator will be superior to the small mutation operator. In domains where an effective state bound setting would be hard to guess the large state bound mutation operator can cover more of the search space faster than the small operator.

4.2 Kalman Filter Results

4.2.1 Test Environment: Grid World

In an effort to verify and extend the results achieved in [17] we used the same environment. The 5x5 Grid World, shown in Figure 4.8, is an environment in which all agents choose an action at each time step and are given a reward for entering a state. At each time step an agent can choose to move in one of four cardinal directions, but any movement from state 6 or 16 will always move the agent to state 10 and 18, respectively. Most states provide 0 reward but the two states that automatically move the agent provide a reward of 20 and 10, respectively. Each agent’s perception is limited to only its own state and is not affected by the location of the other agents in the world.

Because the global reward is the sum of the individual agent rewards this problem exhibits the factoredness property. The learnability of this environment is dependent on the number of agents contributing to the global utility. When more agents contribute to the global utility the learnability is reduced.

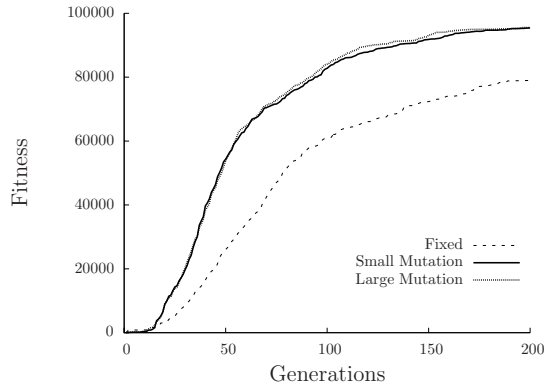


Figure 4.5: This graph shows the average performance over all initial settings of the state bound from 10 to 100 on the double pole balance problem.

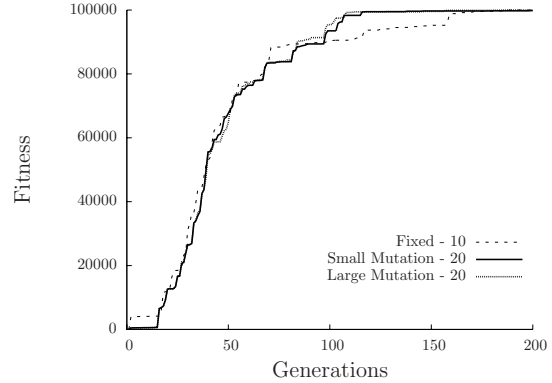


Figure 4.6: This graph shows the best performance observed for any single state bound setting. For Fixed the best state bound setting was 10. 20 for Small and Large mutation.

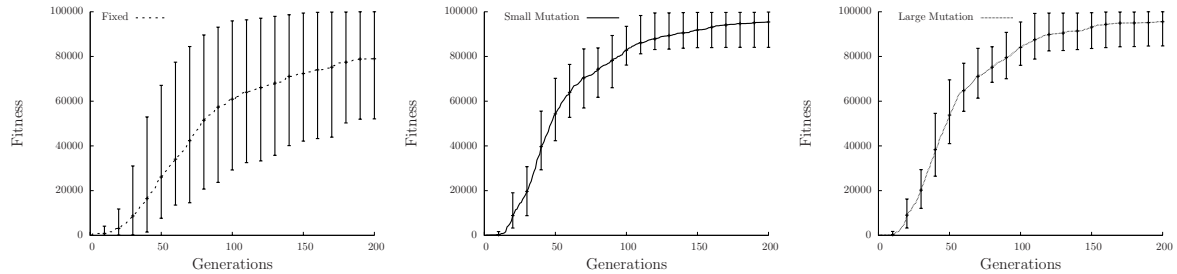


Figure 4.7: These graphs show the average performance for all the methods individually as well as bars that show their maximum and minimum observed performance. Left is Fixed, middle is Small mutations, and right is Large mutation.

In order to analyze the interactions between the Kalman Filter and the reinforcement learners we tried a number of different settings for most of the available parameters while leaving the others invariant. We explored the effects of varying the number of agents, learning rate, epsilon, Kalman variance.

4.2.2 Results

Our initial analysis found similar results to [17] providing effective local reward estimations for 10 agents, an environment that would otherwise lead to poor performance using reinforcement learning based on the global utility. In the following graphs each line represents the local utility of one agent in the simulation.

The effects of interference can be seen as all agents decrease in performance even though they seem to have good local policies. We suspect that a reduction in global reward caused by the exploration of one agent reduces the value of the action taken by other agents at the same time. As

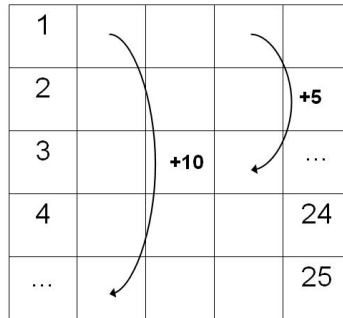


Figure 4.8: 5x5 GridWorld environment

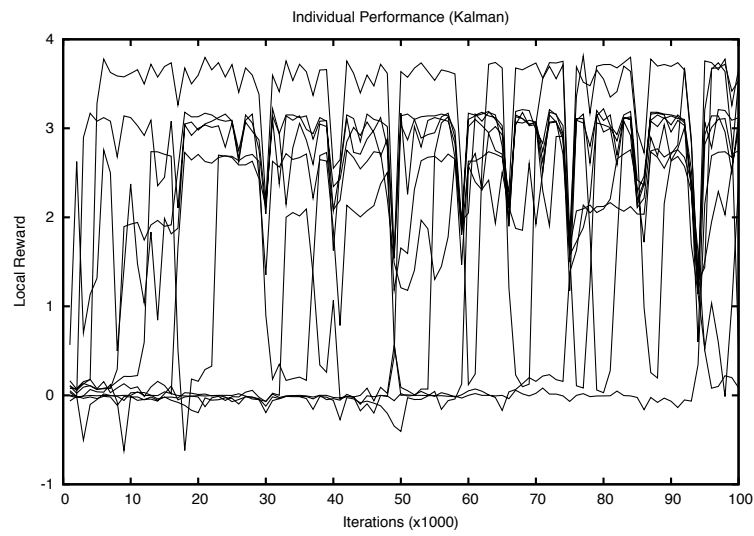


Figure 4.9: Individual agent performance without Kalman filter

a result, agents with a good policy learn to deviate from that policy to try and find a better reward. When multiple agents deviate from their good policy the effect is compounded and all agents in the system decrease in performance and must reacquire a good policy.

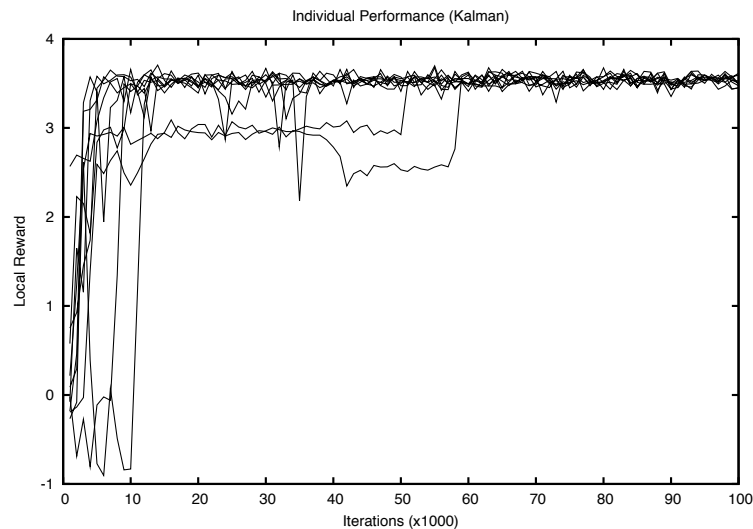


Figure 4.10: Individual agent performance with Kalman filter

With each agent using a Kalman Filter to generate a local reward estimation the effects of interference are minimized and, as a result, all agents are able to achieve better performance. This improvement warranted further exploration of the effects of different settings for learning rate, exploration, number of agents and the Kalman variance that is used as an input to the Kalman Filter. Unless otherwise specified each simulation is an average over 10 runs over a span of 100,000 iterations. The reinforcement learning used a learning rate of 0.90, a discount rate of 0.90, and an epsilon of 0.10 for E-greedy action selection.

4.2.3 Agent Scaling

While the Kalman Filter proved effective for a small number of agents we were interested in analyzing the scalability of this approach. This analysis was done in comparison to a non-Kalman method and also with respect to changes in multiple parameters.

Using a high learning rate that is well-suited for single agent learning, $lr = 0.90$, we found that even adding one additional agent decreased the average performance significantly. In excess of 10 agents the interference and low learnability led to terrible performance in this simple environment.

Since a lower learning rate reduces the effect of interference we repeated the experiment using $lr = 0.3$. As expected, the single agent learned slower but performance was improved when multiple agents were present. However, this improved average performance was still limited after 100,000 iterations and came at the cost of less stability.

In the final experiment each agent is provided with its own Kalman Filter that takes the state and global reward as input at each time step, returning an estimated local reward. Average performance

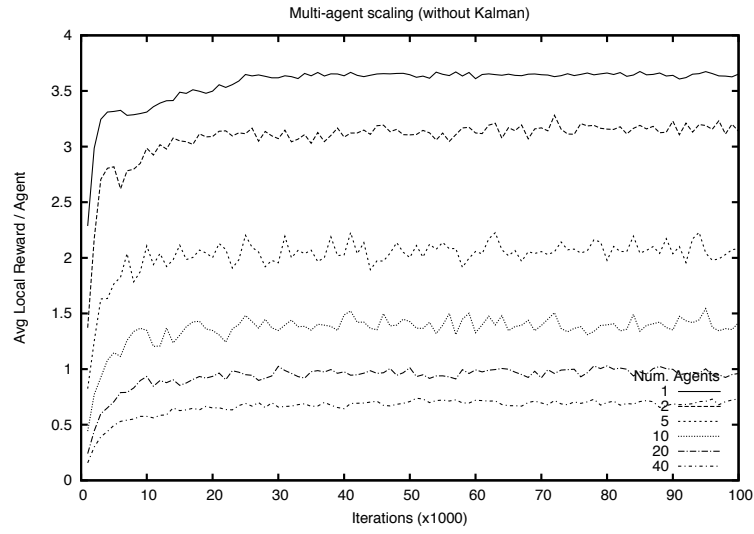


Figure 4.11: Average local reward of all agents with respect to the number of agents in the simulation. No Kalman filter

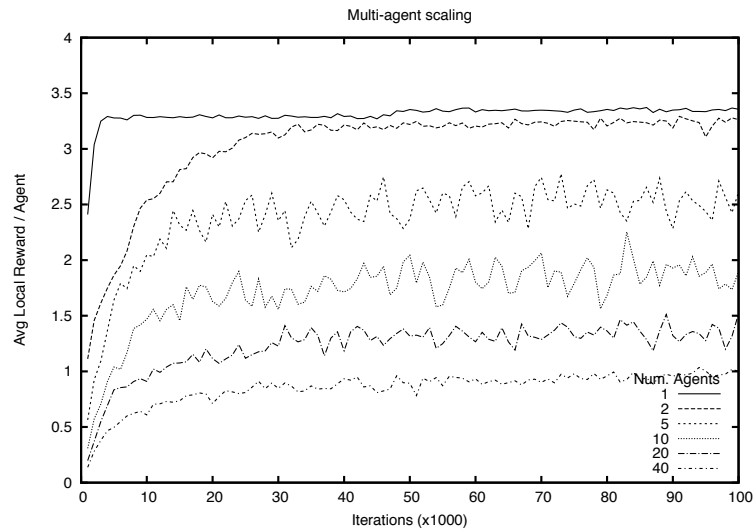


Figure 4.12: Average local reward of all agents using a lower learning rate. No Kalman filter

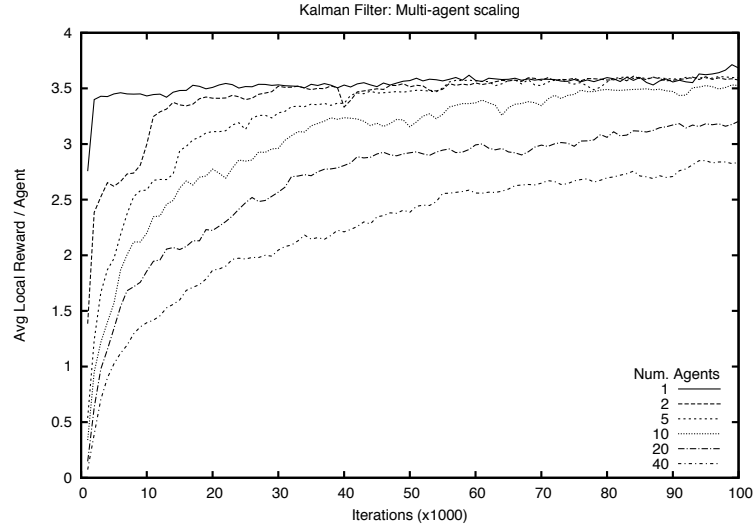


Figure 4.13: Average local reward of all agents using Kalman filter approach

is significantly improved as the number of agents scales to $n = 10$ but eventually the low learnability becomes too significant to overcome. However, even when the number of agents is large the Kalman Filter approach performs much better than Q-learning alone.

4.2.4 Learning Rate

Surprisingly, with 10 agents the learning rate did not seem to have a significant effect on the performance unless a very low value, such as .01 or .10, was used. In fact, an extremely high learning rate of .99 seemed to achieve the best performance. Because the Kalman Filter takes a number of iterations to accurately estimate the local rewards a higher learning rate allows the learner to adapt quickly to the change and capitalize on the accurate estimates as soon as possible.

When running the same experiments with 100 agents, a situation that would be expected to benefit from a lower learning rate, there seems to be little difference in performance using any value between .20 and .99 with .01 providing lower performance and .10 requiring more time to reach the performance of other values.

4.2.5 Exploration

For a fixed number of iterations (100,000 in this case) it is apparent that as the number of agents is scaled up a higher rate of exploration is more effective. This is likely due to the low learnability caused by the number of agents requiring more exploration to discover more combinations of possible agent states which provides a more accurate characterization of the variance for an individual agent for any given state.

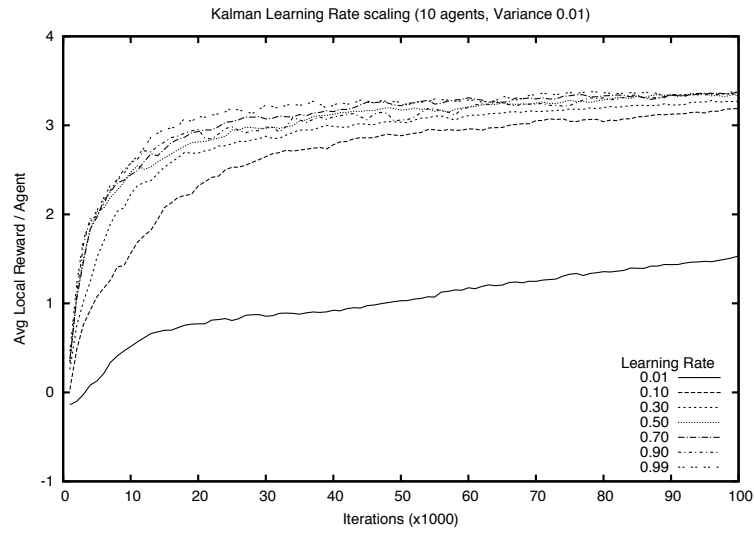


Figure 4.14: Effect of learning rate on performance with 10 agents using Kalman filters

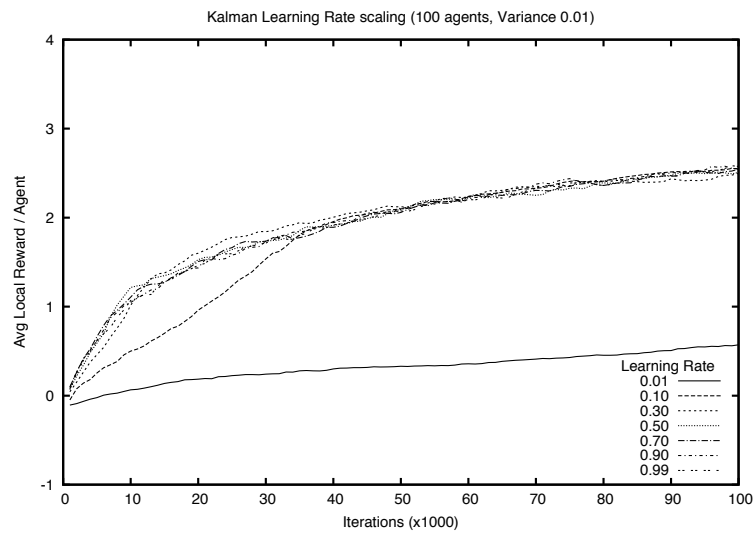


Figure 4.15: Effect of learning rate on performance with 100 agents using Kalman filters

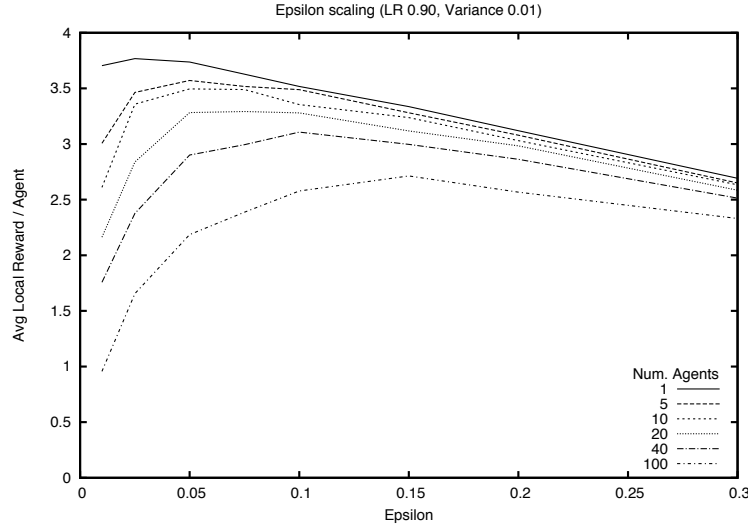


Figure 4.16: Effect of epsilon (exploration rate) on performance with respect to the number of agents

4.2.6 Kalman variance

Aside from setting the initial covariances, the Kalman variance is the only parameter passed into the Kalman Filter and we have confirmed that using a value of 0 causes the filter to fail. After trying many different settings we came to the conclusion that any value $[0.0001, 1.0]$ will suffice and that the filter is not significantly affected by changing this value.

The lack of sensitivity to the Kalman variance was surprising and we initially assumed that it was due to the low number of agents. We were interested to know if there was some correlation between this variance value and the magnitude of the global rewards with a larger number of global utility contributions, such as 100 agents.

Once again, the Kalman variance had no significant effect as long as the value was kept between 0.0 and 1.0. We are still uncertain as to the reason for this peculiarity but based on inspection of the variable affected by the Kalman variance it would seem that it is a multiplicative effect that is dominated by the data itself in most cases.

4.3 AMS - Preliminary Results

In this section we present preliminary results for the Affinity Management System. The main contribution of AMS is its use of *perceived* affinity, which in theory will enable more effective reputation management by exposing more information on the strategies of other agents and improving an AMS agent's standing in an agent society. Unfortunately, at the time this report was written we have not fully explored the use of perceived affinity. Instead we present initial experiments showing AMS is competitive with other trust management approaches without the explicit use of perceived affinity. We also show how perceived affinity values evolve through repeated in-

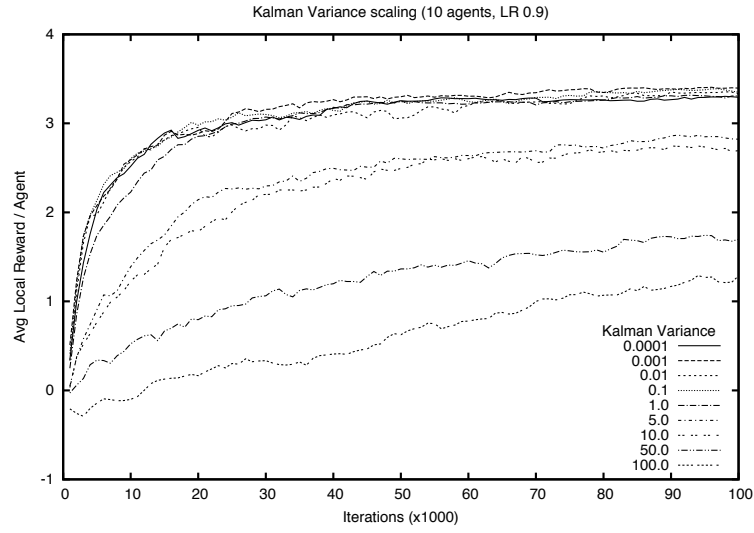


Figure 4.17: Effect of Kalman variance on performance with 10 agents using Kalman filters

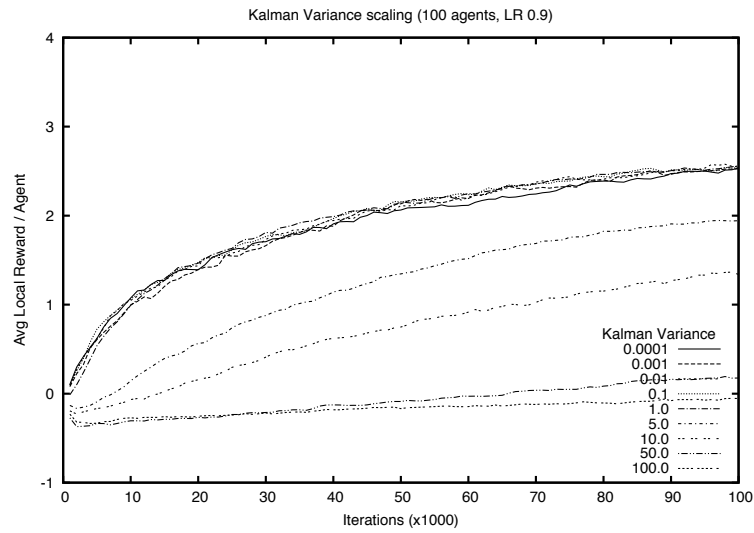


Figure 4.18: Effect of Kalman variance on performance with 100 agents using Kalman filters

teractions with agents utilizing various strategies. This section begins with a discussion of the test environment, proceeds with a discussion of the benchmark agents used in testing, and conclude with the experimental results.

4.3.1 Agent Reputation and Trust testbed

The Agent Reputation and Trust(ART) [11] testbed is a platform developed by students at the University of Texas with the explicit purpose of evaluating distributed agent reputation management schemes. ART simulates a resource constrained environment where agents vie for possession of resources. As such, agents must interact with each other in order to maximize their individual resource coffers. Through agent interaction, ART creates a situation where agents are rewarded on their inherent expertise, as well as their ability to utilize other agents' expertise. The following is a detailed description of ART.

The environment is a simulated art appraisal community where the testbed seeks to mimic established relationships of trust that people establish in an art appraisal environment. There are a couple major relationships that dominate such an environment. The first is the relationship between the client and the art appraiser. The second is the relationship amongst the agents that represent the art appraisers. In the system, each agent represents an individual painting appraiser that can accurately give appraisals on some fraction of the n different eras of artwork in the simulation. The objective of these agents is to maximize the amount of "cash" they receive from successfully appraising pieces of art. Clients, simulated by the environment, are the primary source of cash. There is a fixed number of clients that pay a set amount of cash for an appraisal on a piece of art. Clients seek out appraisers that can evaluate their art works' monetary worth as accurately as possible, while agents seek to maximize their accrued work. The agent appraisers in the environment are competing with each other for a proportion of the customer population. As an agent accurately appraises works of art, their reputation increases amongst the clients proportional to the performance of other agents in the system. However, since no agent is an expert of every era, the agents must contract out work to other agents. The agents can ask and pay one another for opinions on artwork they need help appraising. When working with one another the agents have the choice of helping, not helping, or purposely deceiving the other agent. If an agent is honest in its opinions, the agent will build their reputation amongst the agent community increasing the chance of getting profits from providing secondary opinions. As such, the reputation of agents amongst themselves also comes into play in the accrual of work. What is also important to note, is that the gains that stem from the increases in reputation and trust are still essentially a zero-sum game for the agents in the system. The gain from one agent comes at the expense of every other agent in the system. Hence, there is a real incentive for the agents to not help one another, especially when taking into account how the testbed measures performance. In this particular case, the metric of performance of an agent is the monetary evaluation of their accrued work through the simulation.

The ART platform has come to be accepted by the agent community as one of the de facto validation testbeds for reputation and trust agent algorithms. Top participants from the ART competition have been showcased at the Autonomous Agents and Multi-Agent Systems Conference(AAMAS) annually since 2006 [42]. The competitions have encouraged researchers from

academia and industry to explore various different schemes that have resulted in the development of some exceptional agents.

During the 2007 competition, we participated and the experience of competing, proved to be very beneficial. It revealed insights that have helped our development of trust and reputation agent algorithms. We intend to compete in future competitions with the knowledge gained from the experience. However, what is more important is the leveraging of that experience into the work we are currently proposing. We plan to utilize an agent incorporating a more generalized approach of the domain specific agent that was submitted to the ART competition.

The competitions have provided benefits on two fronts. They have provided us with direct feedback on the performance of AMS and a source of agents built on different reputation management strategies to benchmark AMS against. Some of the top performing agents are available for download as executables from the ART website¹. Unfortunately, given the non-domain specific nature of AMS, the utilization of the top agents from the competition as benchmarks becomes impractical due to their utilization of highly domain specific methods. Therefore, only non-domain specific techniques of the AMS will be focused on.

The ART testbed simulates an environment AMS is designed for, is accepted by the agents community, and provides us with a rich library of alternative strategies to compare AMS against. For those reasons we chose ART for the analysis of AMS.

4.3.2 Benchmark Agents

- **Honest Agent** - This agent is one of the default agents included with the ART testbed and is representative of a completely altruistic agent. The agent always reports its actual expertise for a given era when asked. And when other agents in the system ask an Honest agent for an opinion on a piece of art work it always gives an honest opinion. Honest agent also always believes the opinions it receives from other agents in the system and has no mechanism for seeking the most reliable and honest agents in the society. Honest agents have no trust and/or reputation management system.
- **Cheat Agent** - This agent is one of the default agents included with the ART testbed and is representative of a completely selfish agent. The agent always reports that it is a complete expert in any era when asked. And when other agents in the system ask a Cheat agent for an opinion on a piece of art work it provides an opinion, but does not invest any money in the opinion. Without investing any money in the opinion, the opinions provided are usually considerably off even if the agent is an actual expert in the era. Cheat, like Honest, has no trust and/or reputation management system. The agent also always believes the opinions it receives from other agents in the system and has no mechanism for seeking the most reliable and honest agents in the society.
- **Simple Agent** - This agent is one of the default agents included with the ART testbed and it employs a simple trust mechanism to help it determine the best collaboration partners.

¹<http://www.lips.utexas.edu/art-testbed/>

Simple uses a heuristically derived trust values based on positive and negative interactions with other agents to determine which other agents in the system to cooperate with, trust, and how much to trust them. Trust values are maintained for every agent it interacts with. If the trust value drops below an arbitrary threshold then the agent is no longer trusted. Untrusted agents will not be asked for opinions any longer and if they ask for an opinion the Simple agent's strategy will mimic Cheat. If the trust value exceeds a certain threshold value the agent will become completely trusted. Trusted agents will be asked for opinions on eras they report they are experts in and their opinions will be weighted with the expertise reported. Agents that are not trusted or untrusted are asked for opinions randomly so that a proper trust value for the agent can be calculated. Opinions from these agents are weighted with their reported expertise in the era.

- **UNO Agent** - UNO is the agent that won the 2007 ART competition. It employs complicated heuristic functions designed for the ART competition that help it determine which other agents in the society to cooperate with. The strategy employed is detailed in [25] however the author leaves out the parameters used by the agent to win the competition. In our experiments we utilize a copy of the UNO agent that employs the parameters it used to win the competition.

4.3.3 AMS Benchmark Experiment

The experiment described in this section is a preliminary experiment to evaluate the performance of AMS in the ART testbed. Perceived Affinity is only used for determining the utility of helping another agent. Perceived Affinity is not used to determine the strategy of other agents. This analysis will be pursued in future work. In this experiment we evaluated the AMS agent in the ART competing against four other agents employing different strategies described in the previous section. The final cash balance obtained by the agents at the end of the run was used as the metric to determine the agents' performance. Parameters used for the experiment were 100 time steps, 10 different eras of art, and expertise values were not allowed to change.

Figure 4.19 shows the cash balance accumulated by each agent over the 100 time steps of the run. The figure is only representative of one run of the ART simulation. Many more runs would have to be run before any substantial conclusions can be drawn from the results. However, this result does show that the AMS agent without an enhanced strategy utilizing perceived affinity can be competitive with agent UNO. This is somewhat impressive given that agent UNO was the winner of the 2007 competition and is specifically configured for the environment. The conclusion that we are able to draw from this result is that the current AMS agent has achieved a suitable baseline to determine whether or not the use of perceived affinity will help or hinder an agent once it is properly integrated. Those experiments will be the subject of future work.

The graphs shown in figure 4.20 show the actual and perceived affinity levels the AMS agent derived of the other agents in the simulation. Our belief is that information about the strategy of an agent can be inferred from differences in the actual and perceived affinity levels. If the actual affinity level drops after a drop in perceived affinity, meaning that the AMS agent did something to hurt the other agent's trust of it and the other agent responded negatively, this implies the other

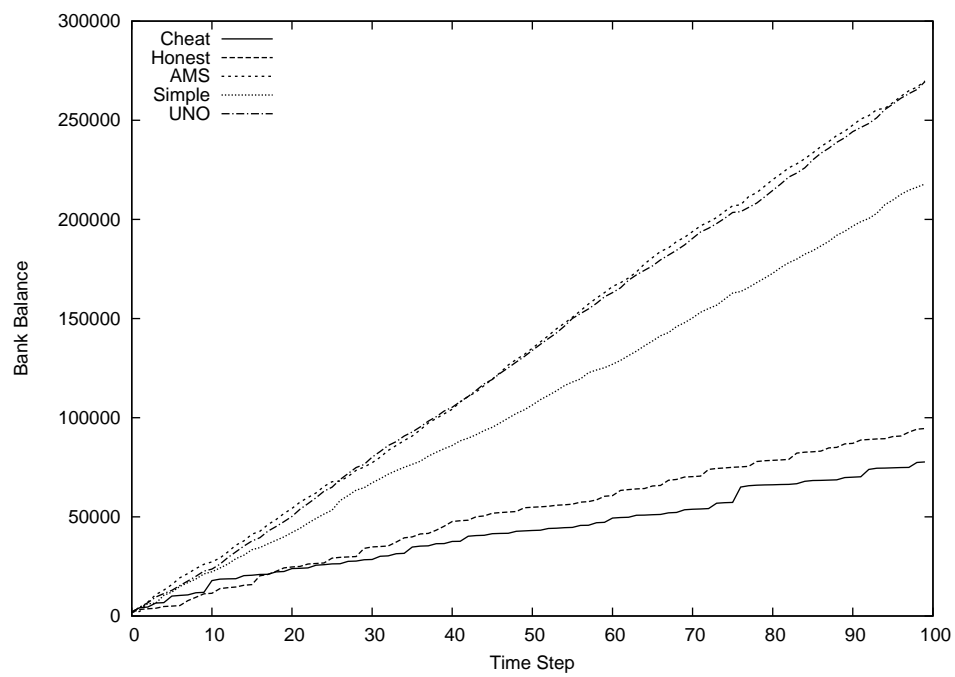


Figure 4.19: This graph shows the bank balance of 5 different agents in a sample run of the ART testbed. On this run the AMS is competitive with agent UNO, the winner of the 2007 competition.

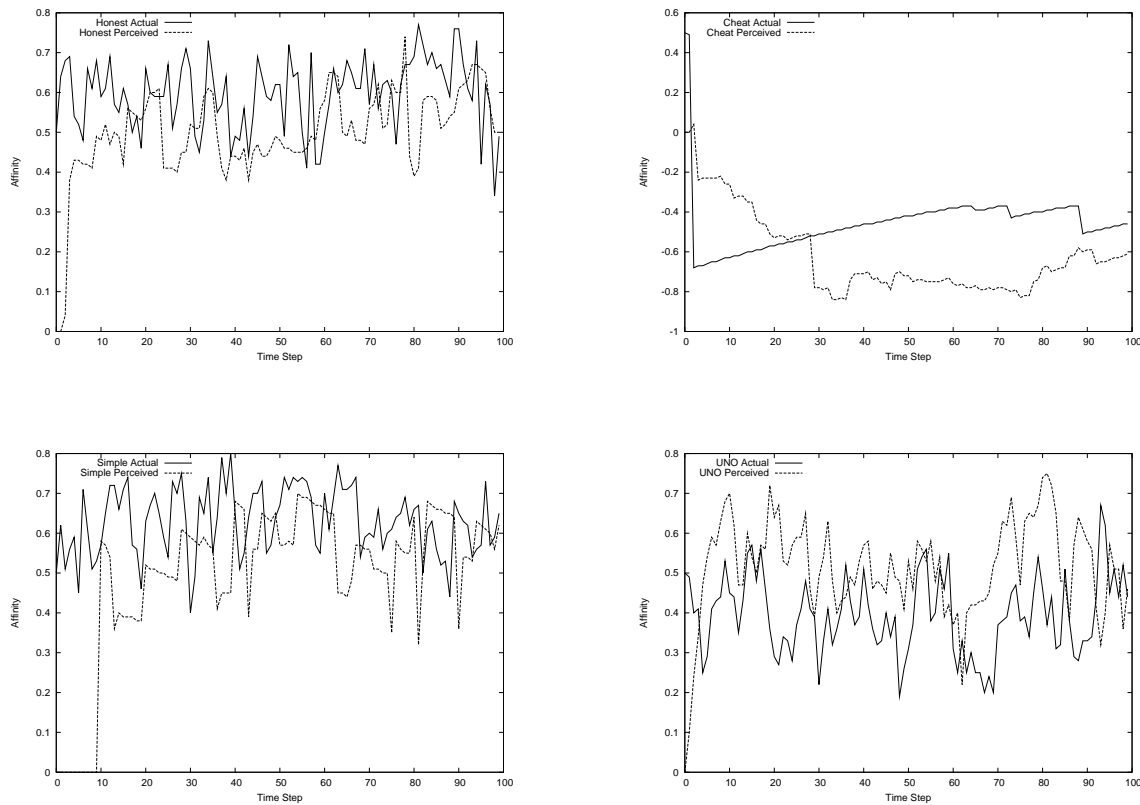


Figure 4.20: These graphs show the actual and perceived affinity values recorded by the AMS agent in the experiment shown in Figure 4.19. (Upper Left) Honest Agent, (Upper Right) Cheat Agent, (Lower Left) Simple Agent, and (Lower Right) UNO Agent.

agent is using a trust mechanism. Agents whose actual affinity level does not change with their perceived affinity level are probably not utilizing a trust mechanism. This theory is not tested here however. These graphs simply serve to show how the two values evolve.

From the graphs it can be seen that the AMS agent quickly learns to distrust the cheat agent and the negative perceived affinity means the AMS agent was also less likely to help the cheat agent during the run. The AMS agent maintains a high level of actual affinity for the honest and simple agents. This is not surprising given that they are designed to be honest and reciprocal. It would be interesting to add a test for retaliatory behavior to the AMS agent's strategy. If the AMS agent purposely performed a negative action toward both Simple and Honest we would most likely have seen a drop in actual affinity for Simple due to a retaliatory reaction whereas honest would not retaliate. Another interesting observation is that the AMS agent usually has a higher perceived affinity value than actual affinity for agent UNO. This means that the AMS agent believes it has done more for UNO than UNO has done for it. This information could be used to alter the strategy with agent UNO so that the cooperation is more equitable.

5 Conclusions

5.1 Automated State Abstraction for Reinforcement Learning

RL-SANE is a powerful new approach for automating the process of state abstraction to enable reinforcement learning to be applied to problems with large state spaces. Our results demonstrate that it is able to solve difficult RL problems more effectively than two standard state-of-the-art approaches, NEAT and CMAC. Neuroevolution provides a mechanism for deriving customized non-linear state abstractions that simplify large and complex state spaces. Combined with temporal differencing learning, the approach is able to learn good policies while also learning effective state abstractions.

We also presented a new method for deriving the state bound parameter, mutation of the state bound. The state bound parameter defines the size of the abstract state space and proper selection is critical to the performance of the algorithm. Previously a proper setting for the parameter had to be discovered through trial and error. The new state bound mutation operators are able to adjust the state bound parameter to a proper value from an initial setting. While the new operators do not eliminate the need for the parameter, the results show that RL-SANE is less sensitive to the initial parameter setting and performs faster overall with the new adaptive methods.

In future efforts we will explore alternative methods for deriving the abstract state space and attempt to apply our approaches to ever more complex domains. RL-SANE utilizes NE to discover NNs that fit an abstract state space of the size and shape that the user defines. The abstract state space in RL-SANE is one dimensional and all the states are of equal size. The NE component spends much of its time manipulating the NN's output to conform with that representation of the state space. It is possible that within a population of NN's there exists NNs that do abstract the space well, but do not conform to the representation of the abstract state space RL-SANE imposes upon it. As a result these networks are abandoned or forced to evolve to fit the abstract state space which takes extra time. It may be possible to identify networks that abstract the state space without conforming to the defined space by evaluating how the NN is clustering outputs. If the clusters form meaningful state abstractions then the approach will work. This will improve the performance of RL-SANE by reducing the complexity of the NE component which is the most expensive component computationally.

RL-SANE has only been applied to academic problems. While these problems are difficult and useful for benchmarking against competing approaches, they do not demonstrate how useful the approach is towards real world problems. In future efforts we will attempt to apply RL-SANE to

more complex and realistic domains.

5.2 Kalman Filter for Multi-Agent Learning

The Kalman Filter has so far proven to be an effective means of estimating local utility in cooperative, model-free, team reward environments, but as the number of agents increases beyond a certain threshold the learnability decreases enough so that the overall performance suffers. However, by changing the value of the exploration parameters with respect to the number of agents in the system the performance can be increased. This motivates further investigation into a means of automatically assigning an exploration rate based on the number of agents (if known) or utilizing a softmax [41] action selection with a cooling factor on the temperature parameter.

Surprisingly, the learning rate and Kalman variance have very little effect on the performance of the agents except in extreme cases. Therefore, it would seem that in this environment a high learning rate, such as the values around $lr = .90$ often used for single learners, would also be effective in the multi-agent case up to $n = 100$. This would indicate that the Kalman Filter is the primary determiner of performance and should be the focus of future work to improve performance using this approach.

In the near future we plan to investigate the effectiveness of the Kalman Filter for local utility estimates in episodic environments in which we expect the speed of learning will decrease. Because the rewards will be provided only at the end of an episode utility estimates will be computed once per iteration of the environment as opposed to being updated at every time step. However, we expect that this approach will still prove effective after a sufficient number of state combinations among the agents has been explored.

In addition we are eager to test this approach in environments that require coordination of multiple agents to achieve a positive utility. In these cases the filter must not only generate an estimate of mapping from local states to rewards but also take into account the states of other agents, increasing the state space significantly. Without this additional state information the utility values will incur additional noise that is dependent on the actions of the other agents with no information to make the distinction between individual and coordinated actions. However, it's entirely possible that the Kalman Filter will recognize an improved average performance when in certain states and cause the learner to prefer those states, increasing the opportunities for positive cooperation.

Because each agent models the contribution of the other agents in the system as noise, some form of communication of local utility estimates could provide a better estimate of the noise term. However, it is still unclear how each agent will weight it's own estimate against the input of others to find a set of values that characterizes the entire global utility received. In addition, because of the scalability issues involved in of broadcasting these estimates there will need to be a mechanism for limiting the communication whenever possible. We have investigated a few possible types of communication, all of which have proved ineffective and led to decreased performance:

- 1) Randomly assign agent pairs at each time step and exchange state information. Use that state information to perform an additional Kalman update using the new state and global utility.

2) Randomly assign agent pairs once at the start of the simulation and perform the same updates as in 1).

3) Same as 1) and 2) but instead of just exchanging state information provide a second communication that returns the estimated local reward for the other agent's state. This estimate replaces the estimate of the originating agent.

The single noise value used by the Kalman Filter may be the limiting factor in environments with low learnability. It's possible that a noise value could be used for each state, but this would likely increase the complexity of the algorithm as well as the number of iterations needed for an acceptable estimate. Therefore, a feasible approach would likely use a combination of local noise values for estimation and a global noise used for computing the covariance of states to noise and noise on itself.

As we explore more complex environments we expect that expanding state spaces will cause some significant scalability issues. Therefore, there is potential for integrating the Kalman approach with state space abstraction techniques, such as CMAC. [3]

5.3 Agent Reputation with Self-modeling

The Affinity Management System is a novel approach to the distributed trust and reputation management problem that includes the concept of *perceived* affinity. Perceived affinity attempts to estimate the trust other agents in a system should have for the agent employing AMS. This estimate, in theory, will provide the AMS agent with information that will help it determine the best collaboration partners, avoid free loaders, and identify the strategies of other agents in the system. In this report we provided a description of AMS and preliminary results of an agent employing AMS in the ART testbed. The results show that AMS is competitive with the best known approach to the ART environment. AMS was able to identify and avoid the free loader in the system and maintain effective collaborations with the other agents in the system. Based on this result we believe that the current AMS agent is a suitable platform for evaluating the benefits of utilizing perceived affinity. Unfortunately, at the time of this writing we have not fully explored the use of perceived affinity. In future work we will derive specific methods and perform experiments to show how perceived affinity can be used to determine the strategies of other agents.

6 References

- [1] Adrian K. Agogino. Unifying temporal and structural credit assignment problems. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 978–985, 2004.
- [2] J. S. Albus. A theory of cerebellar functions. *Mathematical Biosciences*, 10:25–61, 1971.
- [3] J. S. Albus. A new approach to manipulator control: The cerebellar model articulation controller, 1975.
- [4] Robert M. Axelrod. *The evolution of cooperation*. Basic Books, New York, NY, 1984.
- [5] Andrew G. Barto and Sridhar Mahadevan. Recent advances in hierarchical reinforcement learning. 13:2003, 2003.
- [6] R. Bellman. *Adaptive Control Processes: A Guided Tour*. Princeton University Press, 1961.
- [7] Justin A. Boyan and Andrew W. Moore. Generalization in reinforcement learning: Safely approximating the value function. In *Advances in Neural Information Processing Systems 7*, pages 369–376. MIT Press, 1995.
- [8] Anarosa A. F. Brando, Laurent Vercouter, Sara Casare, and Jaime Sichman. Exchanging reputation values among heterogeneous agent reputation models: an experience on art testbed. In *AAMAS '07: Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, pages 1–3, New York, NY, USA, 2007. ACM.
- [9] M. Carreras, P. Ridao, J. Batlle, T. Nicosebici, and Z. Ursulovici. Learning reactive robot behaviors with neural-q learning, 2002.
- [10] Partha Dasgupta. *Trust: Making and Breaking Cooperative Relations*, chapter Trust as a Commodity, pages 49–72. Department of Sociology, University Oxford, 2000. This was originally published in hardcopy in 1988, PDF is 2000 electronic version.
- [11] Karen K. Fullam, Tomas B. Klos, Guillaume Muller, Jordi Sabater, Zvi Topol, K. Suzanne Barber, Jeffrey S. Rosenschein, and Laurent Vercouter. A demonstration of the agent reputation and trust (art): testbed for experimentation and competition. In *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 151–152, New York, NY, USA, 2005. ACM.

- [12] N. Gemelli, R. Wright, and R. Mailer. Asynchronous chess. In *Proceedings of the AAAI Fall Symposium Workshop on Coevolutionary and Coadaptive Systems*. AAAI Press, 2005.
- [13] Faustino Gomez and Risto Miikkulainen. 2-d pole balancing with recurrent evolutionary networks. In *in Proceeding of the International Conference on Artificial Neural Networks (ICANN)*, pages 425–430. Springer, 1998.
- [14] Faustino J. Gomez, Jürgen Schmidhuber, and Risto Miikkulainen. Efficient non-linear control through neuroevolution. In *ECML*, pages 654–662, 2006.
- [15] Faustino John Gomez. *Robust non-linear control through neuroevolution*. PhD thesis, 2003. Supervisor-Miikkulainen, Risto.
- [16] Carlos Guestrin, Michail Lagoudakis, and Ronald Parr. Coordinated reinforcement learning. In *In Proceedings of the ICML-2002 The Nineteenth International Conference on Machine Learning*, pages 227–234, 2002.
- [17] Yu han Chang, Tracey Ho, and Leslie Pack Kaelbling. All learning is local: Multi-agent learning in global reward games. 2003.
- [18] J Holland. Properties of the bucket brigade. *Proceedings of the 1st International Conference on genetic Algorithms and Their Applications*, Jan 1985.
- [19] R. Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82:35–45, 1960.
- [20] Sven Koenig and Reid G. Simmons. Complexity analysis of real-time reinforcement learning. pages 99–105. AAAI Press/MIT Press, 1997.
- [21] Lihong Li, Thomas J. Walsh, and Michael L. Littman. Towards a unified theory of state abstraction for mdps. In *In Proceedings of the Ninth International Symposium on Artificial Intelligence and Mathematics*, pages 531–539, 2006.
- [22] Stephen P. Marsh. *Formalising Trust as a Computational Concept*. PhD thesis, University of Stirling, Apr 1994.
- [23] W.T. Miller III, F.H. Glanz, and L.G. Kraft III. Cmac: an associative neural network alternative to backpropagation. *Proceedings of the IEEE*, 78(10):1561–1567, Oct 1990.
- [24] L. Mui, M. Mohtashemi, and A. Halberstadt. A computational model of trust and reputation for e-businesses. In *HICSS '02: Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS'02)-Volume 7*, page 188, Washington, DC, USA, 2002. IEEE Computer Society.
- [25] Javier Murillo and Vctor Muoz. Agent uno: Winner in the 2007 spanish art testbed competition. In *Workshop on Competitive agents in Agent Reputation and Trust Testbed, The Conference of the Spanish Association for Artificial Intelligence (CAEPIA)*, 2007.

- [26] Jae C. Oh, Nathaniel Gemelli, and Robert Wright. A rationality-based modeling for coalition support. In *HIS '04: Proceedings of the Fourth International Conference on Hybrid Intelligent Systems*, pages 172–177, Washington, DC, USA, 2004. IEEE Computer Society.
- [27] Liviu Panait and Sean Luke. Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems*, 11:2005, 2005.
- [28] Sarvapali D. Ramchurn, Dong Huynh, and Nicholas R. Jennings. Trust in multi-agent systems. *The Knowledge Engineering Review*, 19:2004, 2004.
- [29] Martin Rehak, Milos Gregor, Michal Pechoucek, and Jeffrey M. Bradshaw. Representing context for multiagent trust modeling. In *IAT '06: Proceedings of the IEEE/WIC/ACM international conference on Intelligent Agent Technology*, pages 737–746, Washington, DC, USA, 2006. IEEE Computer Society.
- [30] Jordi Sabater and Carles Sierra. Regret: reputation in gregarious societies. In *AGENTS '01: Proceedings of the fifth international conference on Autonomous agents*, pages 194–195, New York, NY, USA, 2001. ACM.
- [31] Jordi Sabater and Carles Sierra. Review on computational trust and reputation models. *Artificial Intelligence Review*, 24:33–60, 2005.
- [32] Jeff Schneider, Weng-Keen Wong, Andrew Moore, and Martin Riedmiller. Distributed value functions. In *In Proceedings of the Sixteenth International Conference on Machine Learning*, pages 371–378. Morgan Kaufmann, 1999.
- [33] Sandip Sen. Reciprocity: A foundational principle for promoting cooperative behavior among self-interested agents. In *In Proceedings of the Second International Conference on Multiagent Systems*, pages 322–329. AAAI Press, 1996.
- [34] Sandip Sen and Neelima Sajja. Robustness of reputation-based trust: Boolean case. In *In Proceedings of the 1st International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 288–293. ACM Press, 2002.
- [35] Sandip Sen, Ip Sen, Mahendra Sekaran, and John Hale. Learning to coordinate without sharing information. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 426–431, 1994.
- [36] Yoav Shoham, Rob Powers, and Trond Grenager. Multi-agent reinforcement learning: a critical survey. Technical report, 2003.
- [37] Kenneth O. Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. Technical report, Austin, TX, USA, 2001.

- [38] Kenneth O. Stanley and Risto Miikkulainen. Efficient reinforcement learning through evolving neural network topologies. In *GECCO '02: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 569–577, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc.
- [39] Kenneth Owen Stanley. *Efficient evolution of neural networks through complexification*. PhD thesis, The University of Texas at Austin, 2004. Supervisor-Risto P. Miikkulainen.
- [40] Richard Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems*, volume 8, pages 1038–1044. MIT Press, 1996.
- [41] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning)*. The MIT Press, March 1998.
- [42] W. T. L. Teacy, T. D. Huynh, R. K. Dash, N. R. Jennings, M. Luck, and J. Patel. The art of iam: The winning strategy for the 2006 competition. In *The 10th International Workshop on Trust in Agent Societies*, pages 102–111, [”lib/utills:month_13718” not defined] 2007.
- [43] Gerald Tesauro. Temporal difference learning and td-gammon. *Commun. ACM*, 38(3):58–68, 1995.
- [44] Kagan Tumer and David Wolpert. A survey of collectives. In *IN COLLECTIVES AND THE DESIGN OF COMPLEX SYSTEMS*, pages 1–42. Springer, 2004.
- [45] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3-4):279–292, 1992.
- [46] Shimon Whiteson and Peter Stone. Evolutionary function approximation for reinforcement learning. *Journal of Machine Learning Research*, 7:877–917, May 2006.
- [47] Shimon Whiteson, Peter Stone, Kenneth O. Stanley, Risto Miikkulainen, and Nate Kohl. Automatic feature selection via neuroevolution. In *Proceedings of the Genetic and Evolutionary Computation Conference*, June 2005.
- [48] Shimon Whiteson, Matthew E. Taylor, and Peter Stone. Adaptive tile coding for value function approximation. Technical report, University of Texas at Austin, 2007.
- [49] Robert Wright and Nathaniel Gemelli. State aggregation for reinforcement learning using neuroevolution. In *In Proc. 1st International Conference on Agents and Artificial Intelligence (ICAART. INSTICC)*, 2009.
- [50] Yu Zheng, Siwei Luo, and Ziang Lv. Control double inverted pendulum by reinforcement learning with double cmac network. *Pattern Recognition, International Conference on*, 4:639–642, 2006.

7 Symbols, Abbreviations and Acronyms

AAMAS - Autonomous Agents and Multi-Agent Systems Conference
AChess - Asynchronous Chess
AF - Air Force
AMS - Affinity Management System
ANN - Artificial Neural Network
ART - Agent Reputation and Trust testbed
C2 - Command and Control
CMAC - Cerebellar Model Articulation Controller
COIN - Collective INtelligence
E-Greedy - Epsilon Greedy
GA - Genetic Algorithm
KF - Kalman Filter
MAS - Multi-Agent System
MDP - Markov Decision Process
ML - Machine Learning
NE - NeuroEvolution
NEAT - NeuroEvolution of Augmenting Topologies
NN - Neural Network
NP-Hard - Non-polynomial hard
P2P - Peer to peer
PRM - Probabilistic Reciprocity Model
RL - Reinforcement Learning
RL-SANE - Reinforcement Learning using State Aggregation via NeuroEvolution
TWEANN - Topology & Weight Evolving Artificial Neural Network